IMAGE RETRIEVAL AND GEOLOCALIZATION WITH DEEP LEARNING

A Dissertation
Presented to
The Academic Faculty

By

Nam Vo

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

May 2019

IMAGE RETRIEVAL AND GEOLOCALIZATION WITH DEEP LEARNING

Approved by:

Dr. James Hays, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Irfan Essa
School of Interactive Computing
Georgia Institute of Technology

Dr. James Rehg
School of Interactive Computing
Georgia Institute of Technology

Dr. Nathan Jacobs
Department of Computer Science
University of Kentucky

Dr. Aaron Bobick
School of Engineering and Applied Science
Washington University in St. Louis

Date Approved: December 11, 2018

A great quote to start the thesis

George P. Burdell

Dedicated to families and friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In recent years, the recognition community has broadened its focus beyond object categorization to the understanding of a litany of object, scene, material, or 3D attributes. One of the most important attributes of an image is *geolocation* – if we know the location of a photo, we trivially know hundreds of additional attributes (any attribute for which a map exists, e.g. population density, average temperature, crime rate, elevation, distance to a McDonald's, etc.). Knowing the location of an image is also a common photo forensics task. For example, the mobile app *TraffickCam* collects hotel room images to locate incidents of abuse. Unlike many computer vision tasks, computational systems typically exceed the performance of humans at image geolocalization because it is hard for humans to have an accurate visual model of the entire world.

In this proposal, we will apply latest advancement of deep learning to the image geolocalization problem.

Formulated as a learning task, one can either try to directly infer the location from query image, or make an indirect prediction by matching with reference images of known locations. The later has the advantages of being easily extendable (by indexing more images or adding new locations) and has been adopted to various localization or matching/ranking problem.

The community has recently found deep learning techniques to outperform hand-crafted features for matching and ranking images. Moreover, there is significant room for improvement, especially when tailoring to a specific task.

Thesis statement: Image (geo)localization, especially when formulated as an image ranking/retrieval problem, can be effectively approached by the modern convolutional neural network (CNN), which can learn a meaningful, location revealing representation for

images.

Contributions: We support the above thesis statement using the following contributions:

- **Revisiting IM2GPS in the deep learning era:** Image geolocalization at the world scale (inferring GPS coordinate) is extremely challenging and first approached in IM2GPS [1]. The recent state-of-the-art approach to this problem is a deep image classification approach in which the world is spatially divided into cells and a deep network is trained to predict the correct cell for a given image [2]. We propose to combine this approach with the original IM2GPS approach of image ranking: a query image is matched against a database of geotagged images and the location is inferred from the retrieved set. Through extensive experiments we discover a trade off pattern between geolocalization accuracy at different error tolerance configurations. Such analysis is useful for optimizing the accuracy at a particular localization level (roughly street, city, region, country or continent), allowing a smart system to respond differently to queries such as "In which street was this photo taken?" and "In which region was this photo taken?". Our simple approach achieves state-of-the-art geolocalization accuracy while also requiring significantly less training data.

- **Localizing and orienting street views using overhead imagery:** We attempt to determine the location and orientation of a (normal ground-level) query image, again in an image ranking setting, but with a reference database of overhead (e.g. satellite) images; since such kind of images are widely-available covering the entire surface of the earth. To this end, we explore several deep CNN architectures for cross-domain matching – Classification, Hybrid, Siamese, and Triplet networks, and propose a new loss function which significantly improves the accuracy of Siamese and Triplet embedding networks.

  This image matching task is challenging not just because of the dramatic viewpoint

difference between ground-level and overhead imagery but because the orientation (i.e. azimuth) of the street views is unknown making correspondence even more difficult. We examine several mechanisms to match in spite of this – training for rotation invariance, sampling possible rotations at query time, and explicitly predicting relative rotation of ground and overhead images with our deep networks. It turns out that explicit orientation supervision also improves location prediction accuracy. Our best performing architectures are roughly 2.5 times as accurate as the commonly used Siamese network baseline.

- **Deep Metric Learning:** is necessary to train a deep network for image ranking/retrieval. Improving image retrieval performance could translate to improvement in the localization task.

  - **Generalization in Deep Metric Learning with respect to embedding layer**: we study Deep Metric Learning systems applying to fine-grained image retrieval applications and analyze the effect of picking different layers in a deep network as the embedding layer. Traditionally the final output (corresponding to the last layer) will be used as the image representation, we discover that it might not be the optimal choice because of overfitting, especially at small scale data; surprisingly the penultimate layer is usually better at generalizing. We conduct a series of ablation studies and demonstrate state of the art result on 3 well studied image retrieval benchmarks.

  - **Deep Metric Learning for Im2GPS:** while image retrieval approach has delivered improvement in the geolocalization task (predicting GPS coordinate). A deep network trained using classification approach might not be optimal in such use case, and training with Metric Learning can be difficult on large scale diverged data. After experimenting with Deep Metric Learning and training several models, we show that they are better compared to classification trained

model at feature extraction, therefore increasing image retrieval and geolocalization performance.

CHAPTER 2

REVISITING IM2GPS IN THE DEEP LEARNING ERA

Image geolocalization, inferring the geographic location of an image, is a challenging computer vision problem with many potential applications. The recent state-of-the-art approach to this problem is a deep image classification approach in which the world is spatially divided into cells and a deep network is trained to predict the correct cell for a given image. We propose to combine this approach with the original Im2GPS approach in which a query image is matched against a database of geotagged images and the location is inferred from the retrieved set. We estimate the geographic location of a query image by applying kernel density estimation to the locations of its nearest neighbors in the reference database. Interestingly, we find that the best features for our retrieval task are derived from networks trained with classification loss even though we do not use a classification approach at test time. Training with classification loss outperforms several deep feature learning methods (e.g. Siamese networks with contrastive of triplet loss) more typical for retrieval applications. Our simple approach achieves state-of-the-art geolocalization accuracy while also requiring significantly less training data.

2.1 Introduction

In recent years, the recognition community has broadened its focus beyond object categorization to the understanding of a litany of object, scene, material, or 3D attributes. One of the most important attributes of an image is *geolocation* – if we know the location of a photo, we trivially know hundreds of additional attributes (any attribute for which a map exists, e.g. population density, average temperature, crime rate, elevation, distance to a McDonald's, etc.). Knowing the location of an image is also a common photo forensics task. For example, the mobile app *TraffickCam* collects hotel room images to locate incidents

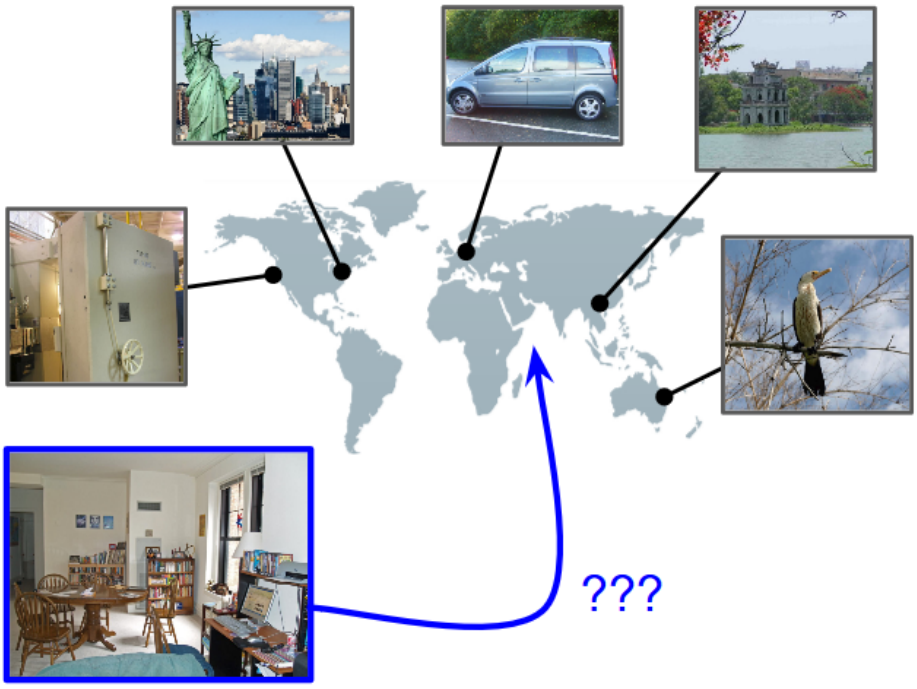

Figure 2.1: This work addresses the image geolocalization problem: given a large set of GPS-tagged images, learn to infer the GPS coordinate of a query image with unknown location.

of abuse. Unlike many computer vision tasks, computational systems typically exceed the performance of humans at image geolocalization because it is hard for humans to have an accurate visual model of the entire world.

Estimating the geolocation of an arbitrary photo is still a challenging task (Figure 2.1). In particular, we examine the task of predicting the location of a single photo given only the image content with no metadata. We consider this task at a global scale and attempt to estimate the GPS coordinates for any query image. For this task, localization can be considered successful if the estimated location is within a specified error threshold. Depending on the application, this threshold could be street level (1km), city level (25km), region level (200km), country level (750km), or continent level (2500km). We adopt these five levels of granularity from prior work and examine the performance of geolocalization strategies at these error thresholds.

One natural approach to the image geolocalization task would be to to treat it like an *instance retrieval* task and match local features from the query image (and perhaps their

geometric layout) to a reference database of images with known locations [3]. Such approaches work well if (1) there are images in the reference database with a field of view that significantly overlaps with that of the query image and (2) if the content of the query image is well suited to local feature matching (i.e., it has distinctive man-made or geological features). Unfortunately, this is often not the case, especially for query images away from tourist destinations and dense urban areas. Therefore, it is necessary to infer location without requiring direct local-feature matching. In these cases, image geolocalization is similar to *scene classification* or *scene attribute estimation* in that a system needs to achieve a higher-level, more qualitative understanding of an image, e.g. recognizing that buildings are typical of Greek islands even though this particular island isn't in the reference database.

Im2GPS [1, 4] introduced the global geolocalization problem and used hand-crafted features from the *instance recognition* and *scene classification* literature jointly to retrieve nearest neighbors in a database of 6 million geotagged images. Im2GPS found that roughly half of successful geolocalizations are instance level matches whereas half are more qualitative matches based on shared scene attributes (geology, architecture, land cover, etc.).

More recently, PlaNet [2] formulates image geolocalization as a classification task. This is done by mapping the GPS coordinate (a pair of real numbers) to a discrete class label by dividing the surface of the earth into distinct regions. PlaNet proposes a deep convolutional neural network to estimate a probability distribution over regions from raw pixel values. PlaNet not only significantly outperforms Im2GPS in terms of accuracy, it is also dramatically faster since it requires only a forward pass through a deep network instead of a nearest neighbor search through millions of image features.

Is the deep image classification formulation of PlaNet the best approach to geolocalization (as it seems to be for most other scene understanding tasks)? There are two reasons to suspect it might not be ideal – first, discretizing the Earth's surface is lossy since we are ultimately interested in a real-valued location estimate (potentially expressed through GPS

coordinates). Second, and more limiting, is that a single deep network, even with tens of millions of parameters, will struggle to memorize the visual appearance of the entire Earth. An effective deep network needs to learn to do both *instance* matching and more *qualitative* scene understanding. Can contemporary deep networks implicitly 'memorize' tens of millions of photographic features necessary for the instance matching?

In this work, we adopt the retrieval approach of Im2GPS but pair it with deep feature learning as in PlaNet. We outperform PlaNet by a significant margin – **47.7% accuracy vs 37.6% for PlaNet** on the Im2GPS test set with a 200km threshold . Interestingly, while we approach geolocalization as a retrieval task with learned deep features, we don't see a benefit to using embedding formulations (e.g. Siamese networks with contrastive or triplet loss) typical for retrieval tasks. Our best performance comes from training a classification network, in the spirit of PlaNet, and using its intermediate activations as our image feature.

The contributions of this study are:

- We significantly improve the state-of-the-art accuracy for global image geolocalization. Our increase in accuracy is similar in magnitude to that achieved by PlaNet [2] over the hand-crafted retrieval approach of Im2GPS. We achieve this with as little as 5% of the training data used by PlaNet, and increase the gap further while using 28% as much reference data.
- Our increase in accuracy comes from changing the formulation from classification to retrieval. The benefit of retrieval in this setting is a reflection of the geolocalization problem and the nature of current deep models – the visual world is too complex for a deep model to memorize, but a retrieval approach does so trivially.
- We investigate different strategies for learning a deep feature embedding for geolocalization. Surprisingly, deep feature learning methods typically used for retrieval applications do not outperform training with a classification loss. For classification-based localization, we find that different discretization strategies also have a significant impact.

- Through extensive experimentation, we find that some training procedures lead to higher accuracy at the street scale (1km) and others at the country scale (750km). We observe a trade off between fine-scale and coarse-scale performance, the regimes traditionally approached with instance-level matching methods and scene classification methods, respectively.

Related works are discussed in the next section. We describe image geolocalization system designs in Section 3. Experiments and analysis are reported in Section 4 and we conclude in Section 5.

2.2 Related Work

Recent years have seen a dramatic expansion of deep learning methods for scene understanding tasks [5]. Deep learning has been applied successfully to location prediction [2] and other tasks related to our problem: predicting scene type [6], perceptual attributes [7] such as safety, liveliness and geo-informative attributes [8] like GDP, elevation.

Image retrieval using learned, deep representations is useful to a wide range of tasks such as product ranking [9, 10], sketch based image retrieval [11], face recognition [12, 13], cross-view localization [14, 15, 16] and scene retrieval [17, 18, 19, 20]. Distance metric learning (DML) is usually employed with a deep network, most commonly using the contrastive loss [21] or triplet ranking (hinge loss) [22, 23, 24]. New loss functions and example mining strategies have been being proposed as they play important role in the learning process [25, 26, 19, 16].

We are studying image retrieval geolocalization which has overlap with *instance-level scene retrieval* [17, 18, 19, 20]; Since this line of work mostly focuses on instance-level matching, benchmarks designed for this task consist of popular scenes or landmarks [27, 28, 29] and similarity between matched images are visually recognizable (by humans or geometric verification). In this regime, with manual labeling and/or clever example mining, it is beneficial to apply distance metric learning. Techniques such as geometry verification

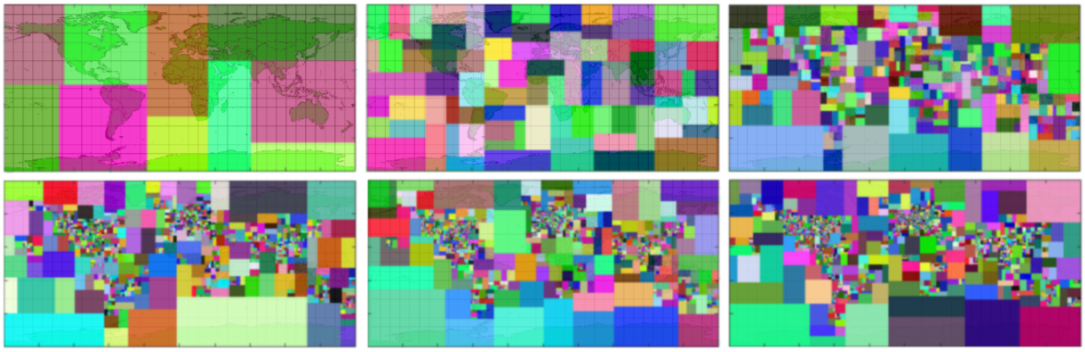

Figure 2.2: We study six schemes for discretizing geographic location. These vary from coarse to fine (10, 80, 359, 1060, 1693 and 7011 regions respectively).

or query expansion typically improve instance retrieval mAP, but these techniques are less useful when geolocalizing scenes that do not have instance matches.

Many previous works on image localization are at limited spatial scale (urban areas) or on special class of images (landmarks, streetview) [3, 30, 14, 16, 31, 32, 33]. Many approaches make used of aerial imagery for localization [34, 34, 35, 32]. In [14, 16, 15], images of the same scene from the ground viewpoint and overhead viewpoint are embedded in the same feature space through deep learning DML; the resulting system then does localization by image retrieval using reference database of aerial images. Also related, to match aerial images across wide baselines, Altwaijry et al. [31] propose a deep attentive architecture to classify whether two views match.

Image geolocalization at planet scale is challenging and less studied – only Im2GPS [1, 4] and PlaNet [2] aim for global coverage. These are the most closely related works on we build on both.

2.3 Image Geolocalization using Deep Learning

Given a large training data of images with GPS labels, we examine two deep learning approaches for geolocalization. For both cases we use the same architecture shown in Figure 2.3 which has been popular for landmark recognition [18, 19, 20].

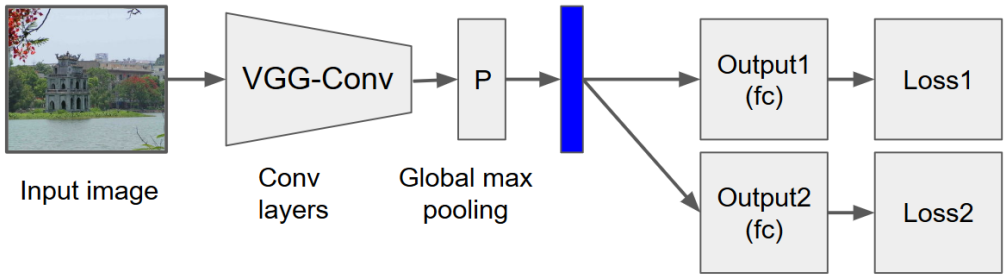

Figure 2.3: Our proposed CNN architecture consists of the convolutional layers of the VGG-16 network [36] followed by a global max pooling layer. Depending on the task, we append to this an output layer and the corresponding loss layer. For classification, we use a fully connected layer and Softmax-CrossEntropy loss, for retrieval, we use a DML loss.

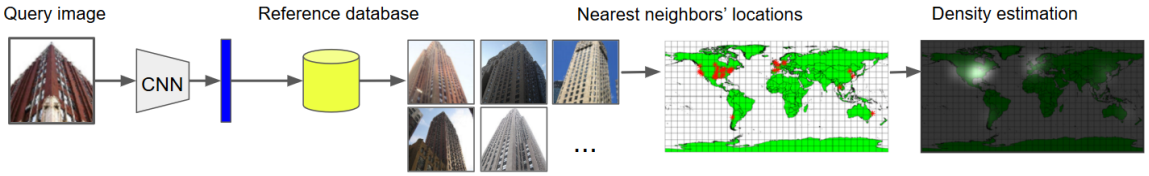

Figure 2.4: A visual overview of our image-retrieval approach to image geolocalization. We extract a feature from our CNN, find nearby neighbors in feature space, and estimate the GPS coordinate using either the top NN or the density.

2.3.1 Geolocalization by classification

One approach is to formulate geolocalization as a classification problem [2]: the GPS label is converted to class label by quantizing all GPS labels to a fixed number of classes, so that each class represents a physical region in the real world. The classification result then can be converted back to the GPS coordinate of the corresponding region.

PlaNet[2] divides the Earth into a set of geographical cells based on image density. We derive a similar adaptive scheme: starting with a single cell of the entire world, repeatedly divide each cells along latitude or longitude whichever side is bigger (either evenly or randomly) until the number of images in each cell is smaller than a threshold t_{img} or the physical area is smaller than a threshold t_{area}; these parameters define how fine the partitioning is.

To predict the location as precisely as possible, one would prefer a fine-grained parti-

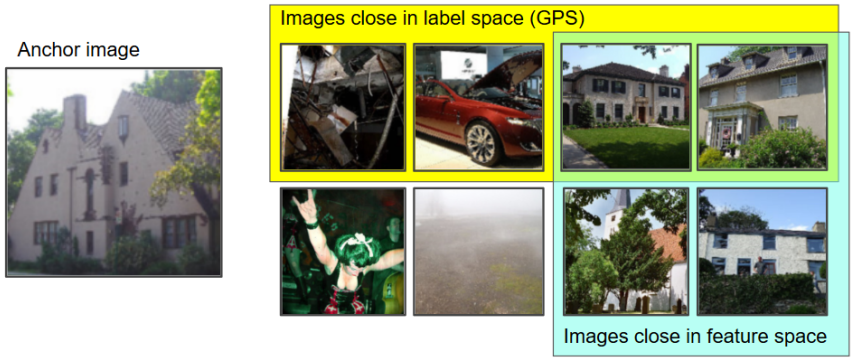

Figure 2.5: When performing distance metric learning, we sample images based on their distance, either in label space (geographic distance) or feature space, to an anchor image. Some example images that are close/far from an anchor image in the label space/feature space.

tioning (for example [2]'s partitioning has 26,263 cells). However we should take into account the training data's size, the learning model's capacity and especially the localization error tolerance. We investigate 6 different partitionings shown in Figure 2.2. Admittedly these choices are somewhat arbitrary, as we do not directly control the number of cells, nor do we try to "optimize" each partitioning. We used similar parameter to [2] to obtain a fine grained partitioning (though [2]'s data is $\sim$14 times bigger so they still have $\sqrt{14}$ times more cells); then we loosen the thresholds to obtain the other 5 coarser partitionings.

Multiple class labeling: We investigate the effect of using multiple partitionings simultaneously. The motivation is that different proximity information is preserved at different levels of granularity (and not the others). Moreover classification results at multiple coarse partitionings can be combined to produce a more fine grained prediction. Therefore we experiment with training multiple classification losses as these tasks are heavily correlated and benefit each other.

2.3.2 Geolocalization by image retrieval

This approach looks up images that are similar to the query images and makes use of the known locations of those images [1]. This requires learning a representation for comparing images (for which we will use deep learning) and indexing a large reference database.

To learn such a representation, we employ distance metric learning (ranking/triple hinge loss, contrastive loss and similar loss functions) which requires pairs of images labeled 'similar' or 'different'. When not available, such labeling can be automatically generated using geometry verification [20, 19] or class labels [26, 25]. In our case, we make use of the class label described in the previous section or directly threshold the GPS distance between the 2 images. Similar to [18], we can also match images that are not only close in the GPS label space but also close in the current feature space. Even so, with the data we are dealing with, this supervision is very weak in the sense that matched images (taken at the same location/region) are most likely not of the same or even similar scene/object (Figure 2.5).

After training we use the CNN as a feature extractor and index a large dataset of reference image features. At test time, we look up the nearest neighbor (NN) of the query image in the feature space using approximate NN search and output its location (Figure 2.4). This approach works based on the assumption that, after learning, images close in the feature space are likely to be close in the label (GPS coordinate) space too.

k-NN density estimation: we can make use of the top k NN instead of only 1. We perform weighted kernel density estimation using each NN as a Gaussian kernel, the density at a point x in GPS coordinate space can be written as:

$$f(x) = \sum_{i=1}^{k} w_i N(x; x_i, \sigma^2 I) \tag{2.1}$$

Where x_i is the GPS coordinate of the i-th NN, we also weight each NN $w_i = s_i^m$ depending on its similarity score s_i (defined to be the inverse of the distance between the query image's feature and the reference image's feature). The point with highest density is chosen as output.

Note that as k decrease, m increase or σ decreases, this output becomes the NN. These parameters can be optimized: bigger reference data allows bigger k and looser error thresh-

old allows bigger σ. Given our dataset (described in the next section) We choose $m = 10$, $k = 100$ through validation (these parameters were not precisely tuned) and experimentally manipulate σ.

2.4 Experiments

Training data: We use the Im2GPS dataset from [1]. It consists of more than 6 million images collected from Flickr that are tagged with countries or states' name and also have GPS coordinates. This data is used for GPS quantization (Figure 2.2), training deep networks, and as retrieval reference database.

Testing data: for analysis, we construct 2 test sets; we make sure that no image from training and test data come from the same photographer.

- **Im2GPS3k**: 3000 images from Im2GPS. Note that this is different from the Im2GPS test set [1].
- **YFCC4k**: 4000 random images from the YFCC100m dataset [37]. Since it is designed for general computer vision purpose, its image distribution is different from Im2GPS making this test set more challenging.

Training for classification: we train the following networks:

- **L**one: We trained a network with a single classification loss corresponding to the most fine grained partition (7011 classes). This can be considered an analog of PlaNet [2] at smaller scale. We also train another version **L2** for the 359 ways classification loss.
- **M**ulti: We train another classification network with 6 different losses corresponding to 6 partitions scheme described in section 3.1. Hence this network produces 6 localization outputs . We'll treat these outputs independently and evaluate the performance of each of them.

Training for retrieval: we fine-tune the model L with ranking loss (triplet hinge loss) to learn a better representation, resulting in a **R**anking network. To do localization by retrieval, we experiment with different networks as feature extractor: the classification networks (L and M) and the ranking network (R).

We also evaluate two other publicly available state-of-the-art models, NetVLAD[18] and Siamac[19], which have similar architecture (VGG-conv layers), but different training data (weakly supervised Google streetview time machine and SfM landmark images hard example mining), global pooling layer (NetVLAD and R-Max) and loss function (triplet hinge loss and contrastive loss). Different from our approach, these models have an additional fully-connected layer for PCA. Features from all models are L2-normalized when used for retrieval.

Notation: we will use $[Model]Approach$ to refer to each method, where $Model$ can be L, L2, M, R, NetVLAD, Siamac described above, and $Approach$ can be C (for classification), NN, kNN (for retrieval). For example [M]311C refers to the 311 way classification output of model M, and [M]NN refer to the NN retrieval approach using model M as feature extractor.

Metric: the geolocalization accuracy is defined as the percentage of test images whose predicted location is within the error threshold from the true location. Similar to [1, 4, 2], 5 error thresholds are used: 1km, 5km, 25km, 750km, 2500km corresponding to 5 levels of localization: street, city, region, country, continent.

Result: Qualitative results are shown in Figures 2.6 and 2.7. Quantitative results on two test sets are shown in Figure 2.8. For comparison we add a simple baseline: always outputting **London**, which is the region with the most images. This baseline is practically the best one can do without looking at the input image; its performance is much better than guessing a random location on the Earth.

We will ensure that our results can be replicated by sharing our datasets, source code, and trained models.

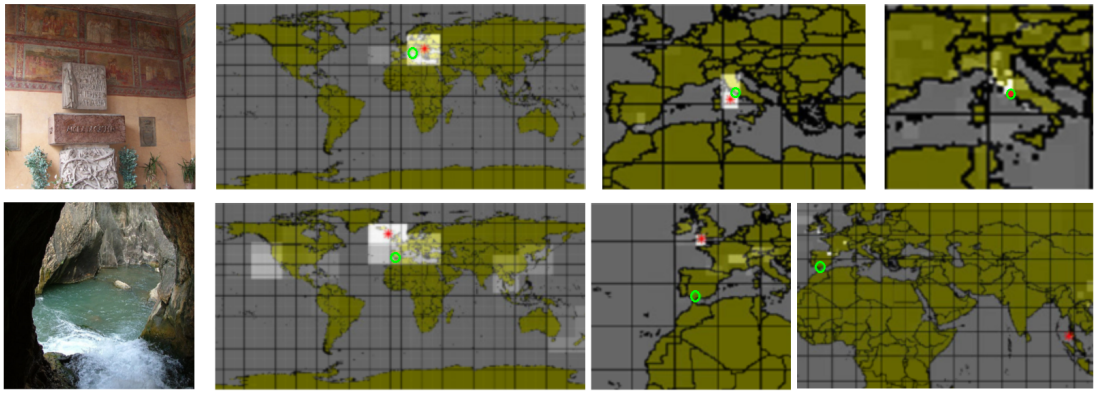

Figure 2.6: Example results of geolocalization by image classification using different partitionings. From left to right: input images, classification result with 80, 1060 and 7011 classes respectively (lighter region means higher probability). Red * denotes the predicted location and green o denotes the true location.

2.4.1 Comparing classification performance

An example output of classification is in Figure 2.6. In the case of less ambiguous image, the network would be able to predict the correct region/cell. Since the center of the region is used, a finer partitioning will lead to a prediction that is closer to the true location (top row). Though in case of the image being very ambiguous, correctly localizing it at coarser level is more likely (bottom row).

As shown in Figure 2.8, the geolocalization accuracy of the 10 way classification output is quite bad, this is mostly because at this scale the Earth is under-divided. We can see that as the partitioning is finer, the localization performance at lower error threshold gets better as expected. The fine-grained classification output (7011C) outperforms others at street and city level.

Most interesting, the geolocalization accuracy at coarse level gets worse if the partitioning is too fine: for example at continent level, the 80C and 359C achieve highest accuracy; At country level, the 359C and 1060C have the advantage. This seems to indicate a trade off between the accuracy at coarse and fine level, which may be a shortcoming of the partitioning in PlaNet [2].

[M]7011C and [L]7011C achieve similar accuracy ([L] is slightly better). However in

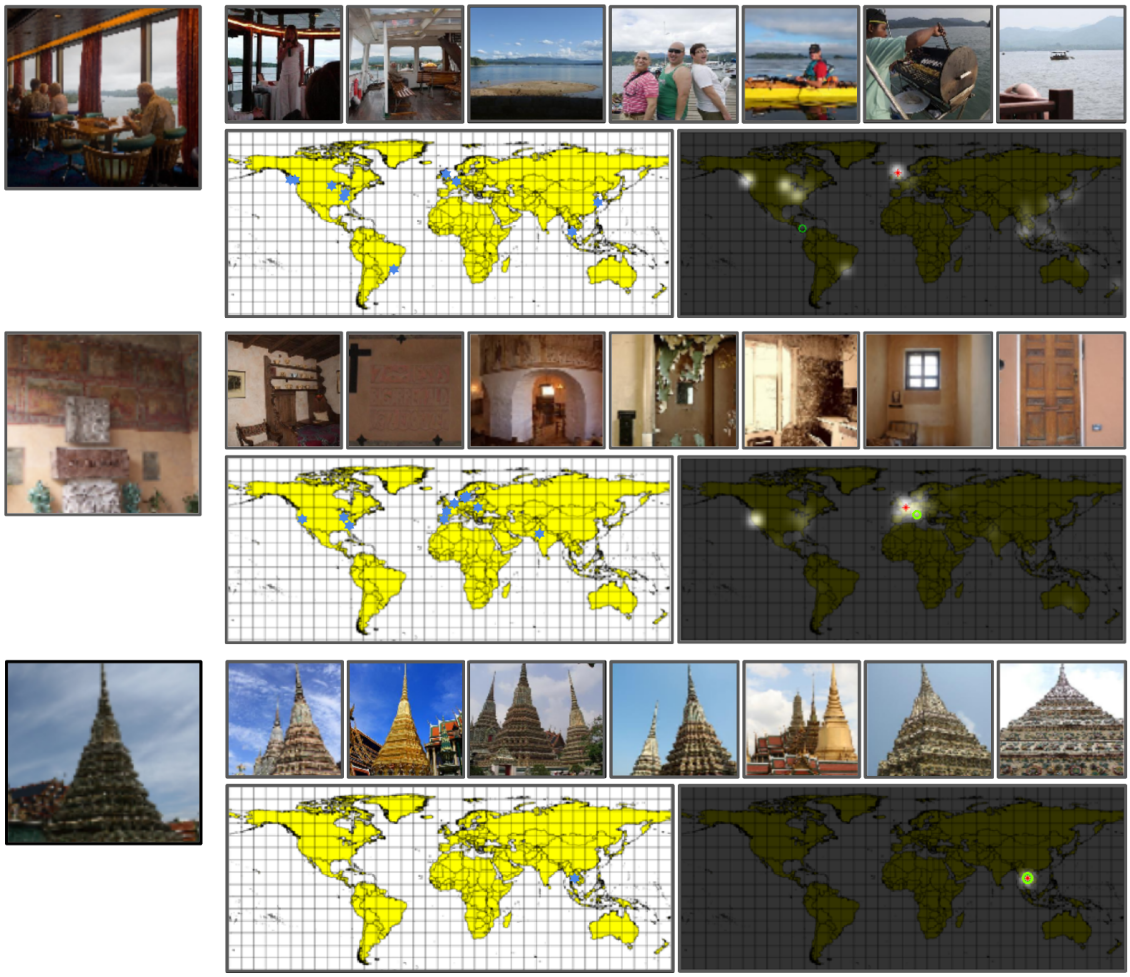

Figure 2.7: Example results of our geolocalization by image retrieval system (kNN, σ=4). Each row shows the input image on the left, the first few NNs on the right, together with their locations (blue *). At the end of the row we show the density result, red * denotes the predicted location and green o denotes the true location.

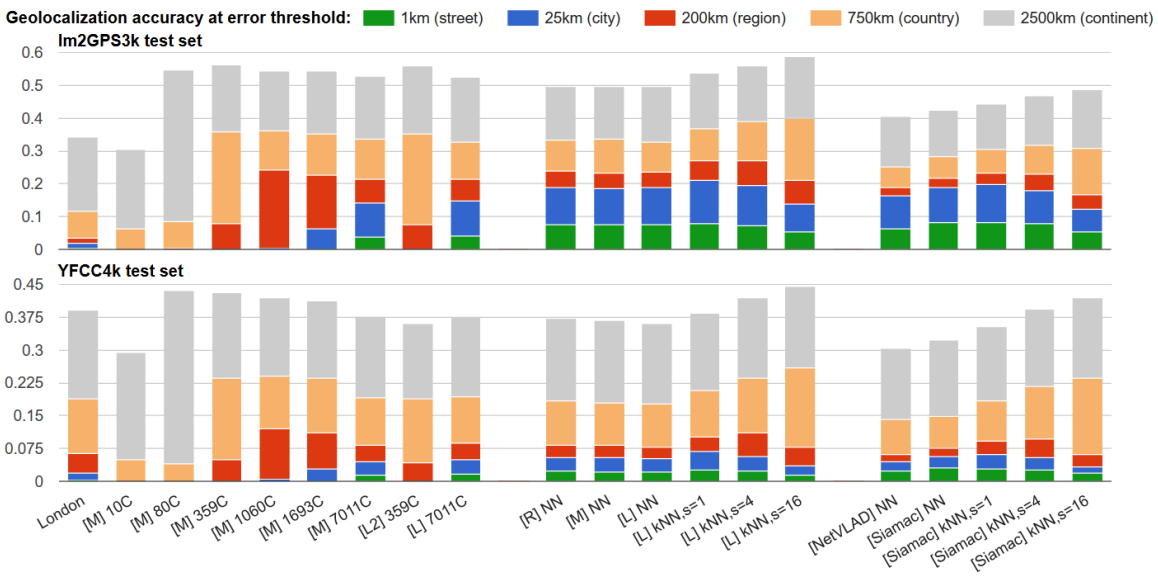

Figure 2.8: Geolocalization accuracy on two test sets. Note that the accuracy is presented as the top of the bars, not the length of each single color.

the case of 359C, [M] is slightly better than [L2]. This suggests that when training with multiple classification losses, the fine-grained one seems to help the coarse one a little, but not vice versa.

2.4.2 Comparing retrieval performance

Figure 2.7 shows example image retrieval results. The NNs are similar scenes to the input image. In the case of landmarks and popular sites, they are usually instance level matches.

As shown in Figure 2.8, with localization by NN image retrieval, all 5 models (R, M, L, NetVLAD, Siamac) perform well and outperform the classification result at street and city level. This makes sense as these successful localizations are likely correct instance-level matches. While classification network can learn the general characteristics of each regions, it doesn't have enough capacity to 'remember' all specific instances, while the retrieval approach 'remembers' this by directly saving all reference features.

Among all 5 models, NetVLAD is the worst. Siamac is the most discriminative at street level. As a trade off, it has slightly lower performance at coarse level (country and

continent). The L and M models are comparable and they perform relatively well even though they are trained for classification. Coarse partitioning classification approaches still have the advantage at country and continent scale.

Finally, using kNN-kernel density estimation improves the accuracy (here we only show [L] and [Siamac] but the changes when using other models are similar); especially at coarse scales (as σ increases) this makes retrieval competitive with the classification approach. However bigger σ can potentially lower the accuracy at fine grained level. Interestingly, we arrive at a similar trade off between fine and coarse geolocalization accuracy.

2.4.3 Training a ranking network with GPS label

Model R (which was fine-tuned from L) doesn't produce a noticeable improvement over L or M (Figure 2.8). In further investigations, we train a dozen versions of R, fine-tuned from different pretrained models and varied the way we sample/mine training examples. In all cases, little progress is observed in term of both training loss and geolocalization performance.

However when using landmark matches from [19] for training instead of Im2GPS data, we observe slight improvement at street level, but worse results at other scales. This is consistent with the fact that Siamac[19] is very good at street level.

Distance metric learning losses like triplet hinge loss seems to be very sensitive to noisy labels. Different from classification loss (where the label for each image is fixed during training), the "target" of each training image keep changing while they are adjusting distance from each other, usually making convergence slower.

We hypothesize that the inter-class ambiguity and intra-class diversity are too large and DML is not able to learn from GPS supervision (Figure 2.5).

2.4.4 Comparing with IM2GPS and PlaNet

On the Im2GPS test set, we can directly compare Im2GPS [1, 4] and PlaNet [2] with two of our models:

- The fine-grained classification network ([L] 7011C). This can be considered the equivalent of Google's PlaNet[2] at smaller scale.
- kNN kernel density estimation retrieval ([L] kNN, σ=4). This can be considered the equivalent of Im2GPS approach [1], but using deep features instead of classical features.

The result is shown in table 2.1. Our classification network outperforms Im2GPS even though it is still not as good as PlaNet. On the other hand, our localization by deep learnt image retrieval method produces even better accuracies. This result highlights the advantage of retrieval approach for fine-grain localization.

Complexity analysis: in term of number of parameters without counting the output layers, PlaNet is 3 times bigger than our 13 layers deep VGG model. Note that PlaNet uses an Inception architecture which has been heavily designed to optimize for complexity [38, 39] (for reference, it is 8 times bigger than 22 layers deep GoogLeNet[38] and 2 times bigger than 42 layers deep InceptionV2[39]). Also PlaNet's training data has more than 90 million images and it takes 2.5 months to train on clusters (approximately 40 years of CPU time). However in term of space complexity, our image retrieval approach requires all reference features be available during testing, not just the deep network. More over, the cost of indexing and perform NN search is not negligible; though indexing needs to be done only once and in our experience the cost of approximate NN search is smaller than that of feature extraction.

Comparing to Im2GPS [1, 4], deep learning feature extraction is orders of magnitudes faster than computing many classical computer vision features. Im2GPS's combined feature has more than 100k dimensions; in [4] lazy learning is done for each query adding

Table 2.1: Performance on Im2GPS test set. (Human* performance is average from 30 mturk workers over 940 trials, so it might not be directly comparable)

	Street	City	Region	Country	Cont.
Threshold (km)	1	25	200	750	2500
Human*			3.8	13.9	39.3
Im2GPS [1]		12.0	15.0	23.0	47.0
Im2GPS [4]	02.5	21.9	32.1	35.4	51.9
PlaNet [2]	08.4	24.5	37.6	53.6	**71.3**
[L] 7011C	06.8	21.9	34.6	49.4	63.7
[L] kNN, σ=4	**12.2**	**33.3**	**44.3**	**57.4**	**71.3**
... 28m database	**14.4**	**33.3**	**47.7**	**61.6**	**73.4**

more time complexity. In contrast, our deep feature with 512 dimensions is suitable for direct comparisons in Euclidean space. Because of this, our kNN kernel density estimation is a more efficient and effective post-processing procedure than the similar kNN mean shift clustering and lazy learning in [4].

2.4.5 Effect of retrieval reference database

One advantage of retrieval approach is that we can simply index more examples to improve the performance. To that end we collect another 22 million GPS-tagged images from the YFCC100m dataset [37], increasing our database size to a total of 28 million images. As shown in table 2.1 (last row), this results in better performance of [L]kNN,σ=4 on the Im2GPS test set.

We vary the reference retrieval database (Im2GPS-6 millions images, YFCC-22 millions and the combined 28 million) and show the geolocalization accuracy in table 2.2. The performance when using YFCC22m is actually no better than when simply using Im2GPS; though the combined database of 28 million images result in an improvement. We attribute this to the fact that the IM2GPS test set and the IM2GPS database come from the same distribution, which makes IM2GPS more useful for referencing. To quantify this, we measure the percentage of IM2GPS images among the top 1, 10, 100, 1000 nearest neighbors result, they are 53.2%, 50.1%, 44.6% and 40.1% respectively, which is quite high given

Table 2.2: Performance on Im2GPS test set based on different retrieval reference database.

Retrieval	Database	Stre.	City	Reg.	Cou.	Cont.
[L] NN	Im2GPS	12.7	33.3	40.9	53.2	71.7
	YFCC22m	12.2	30.4	37.6	51.1	67.1
	Both(28m)	13.9	32.9	40.5	54.4	70.9
[L] kNN $\sigma = 1$	Im2GPS	13.1	36.3	44.3	56.1	70.0
	YFCC22m	12.7	34.2	43.9	55.3	68.8
	Both(28m)	**15.2**	**37.6**	46.0	57.0	69.2
[L] kNN $\sigma = 4$	Im2GPS	12.2	33.3	44.3	57.4	71.3
	YFCC22m	11.8	31.2	42.2	58.7	70.0
	Both(28m)	14.4	33.3	**47.7**	**61.6**	73.4
[L] kNN $\sigma = 16$	Im2GPS	10.6	24.9	35.4	59.5	75.9
	YFCC22m	8.4	19.8	34.6	58.2	74.7
	Both(28m)	11.8	24.9	36.7	60.8	**77.2**

that IM2GPS only constitutes 22.8% of the combined database.

Similar to result on Im2GPS3k and YFCC4k, we can change σ to optimize the accuracy at a localization level (at the expense of the others). If the system is allowed to produce different outputs at different levels, this further outperforms the result in Table 2.1.

2.4.6 Implementation

Here we will provide some more detail about our implementation. We use Caffe framework [40]. We use learning rate 0.01 and reduce it several time during the training, to 0.00001 (when the loss seems to stop improving). Mini-batch size is 32, momentum is 0.9 and weight decay factor is 0.0005.

We use VGG trained on ImageNet [36] as initialization and train a network with the 1060 ways classification for 500k iterations. Then we use this network as initialization for training every other networks (usually just another 100k-200k iterations), we found that this speed up the experiment quite a lot since training every model from scratch or ImageNet initialization take much more time. As shown in Table 2.3, the pretrained ImageNet model ([I]) can be also be used for retrieval, but not as effective as a model trained for geolocalization task ([L]).

Table 2.3: Performance on Im2GPS3k test set.

Method	Model	Stre.	City	Reg.	Cou.	Cont.
NN	[I]	7.4	17.0	19.6	26.8	41.9
	[L]	7.5	18.9	23.5	32.6	49.5
kNN,σ=1	[I]	7.5	18.3	22.5	30.2	45.8
	[L]	7.8	20.9	27.1	36.8	53.8
kNN,σ=4	[I]	7.0	16.8	22.1	31.9	48.7
	[L]	7.2	19.4	26.9	38.9	55.9
kNN,σ=16	[I]	4.4	10.6	15.4	32.2	51.2
	[L]	5.3	13.8	21.2	39.9	58.9

When training with multiple losses, the overall loss will be the weighted sum of all the losses. For [M] model, we use the same weight (1) for all 6 losses.

2.4.7 Feature visualization

We show a t-SNE visualization in Figure 2.9. The feature learnt from GPS supervision seems to be very high level; there's many regions in this visualization with consistent theme such as: sport scene images, people images, beach and sunset images, animal images, landmark type of architecture images, etc. There's a large variety in image appearance within a region.

In Figure 2.10 we look at some dimensions in the output feature space and show the images whose has a high corresponding feature value. Few activation outputs do correspond to some particularly popular landmarks/architecture; while many correspond to certain type of scene or visual features. Some seems to respond to more than one visual features and some might roughly represent higher level location-based semantics. For example row 5 shows pictures of Disney-like castle and Disney's Mickey mouse even-though they are not visually similar.

We show some more nearest neighbors example result in Figure 2.11.

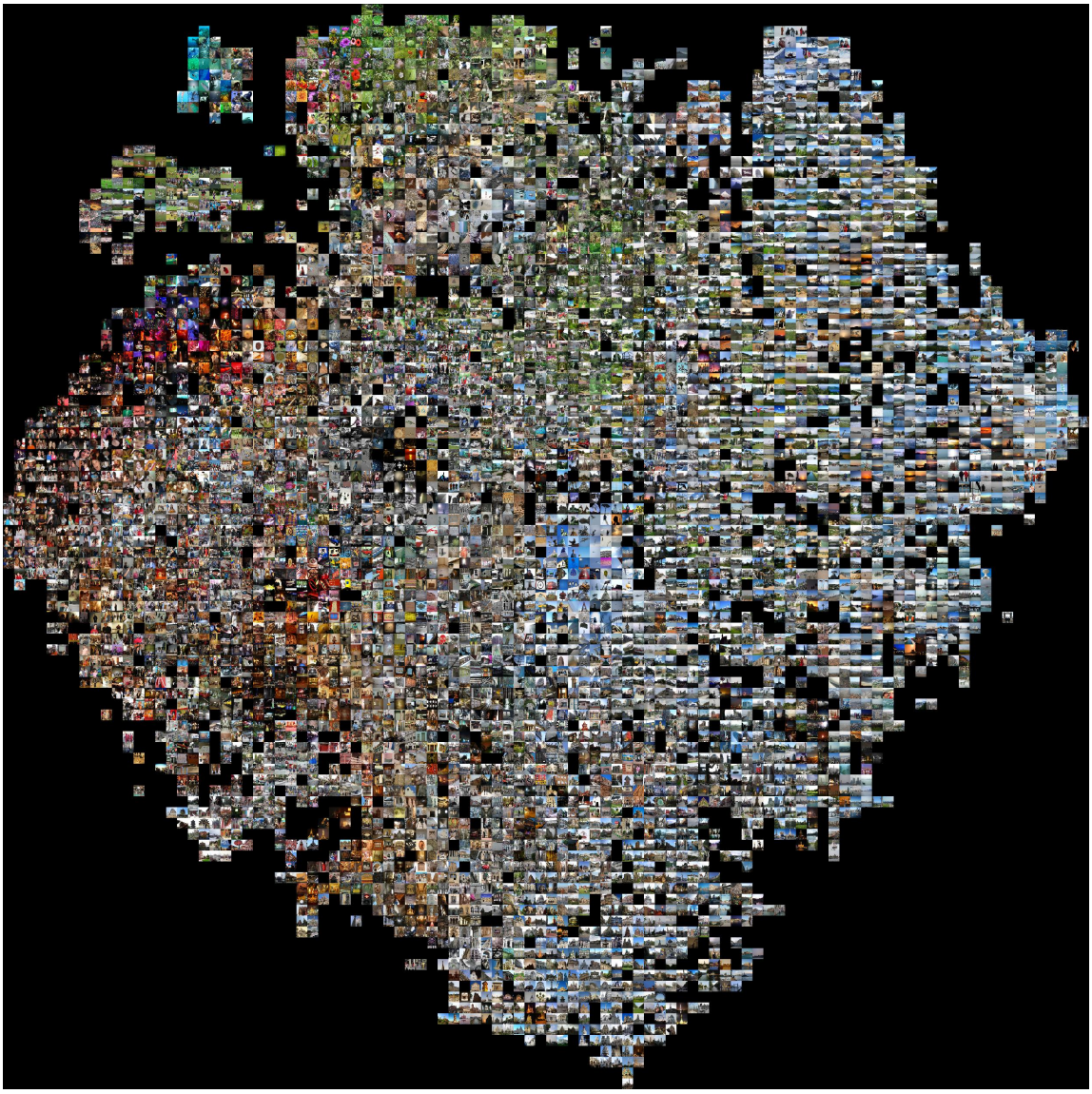

Figure 2.9: t-SNE visualization

Figure 2.10: Each row shows a set of images whose feature has a high value at a particular activiation unit (last layer).

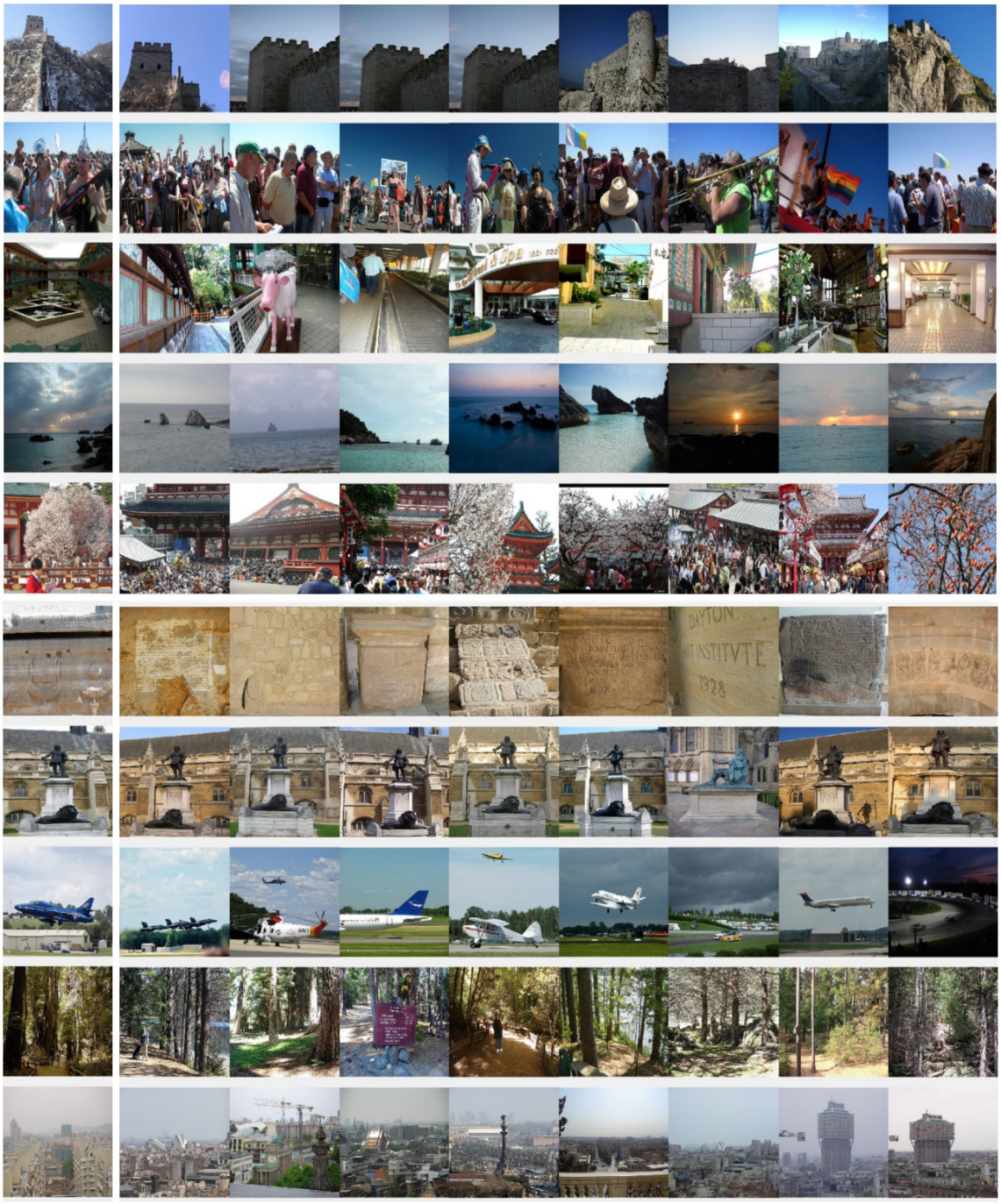

Figure 2.11: Some qualitative near neighbors result: the images on the left column are query, the other on the same row are its NNs.

2.5 Conclusion

We presented a deep learning study on image geolocalization, where we experimented with several settings of image classification and image retrieval approaches adapted to this task. We do not claim technical novelty for any components of this study. Our approaches are relatively simple yet achieve state-of-the-art accuracy. In the end, the best performing models can efficiently and accurately localize at coarse level using classification, and if needed can search for instance matches using retrieval techniques.

The main goal of this chapter is to investigate the effectiveness of deep learning methods for geolocalization. With the newly obtained insights, we think the following lines of future work would be important: (1) we have shown the dependency between partitioning scheme and geolocalization accuracy, which begs the question: what is the best way to partition and how can the partitioning be optimized given a particular error threshold? (2) Are GPS labels too weak a supervision for traditional deep distance metric learning? There is likely an opportunity for better weakly supervised DML to improve the geolocalization.

CHAPTER 3

LOCALIZING AND ORIENTING STREET VIEWS USING OVERHEAD IMAGERY

Different from the previous chapter where we perform image ranking to find images of the same scene or similar scenes, here we explore an alternative: using overhead (e.g. satellite) images as reference. The motivation is that overhead images are widely available covering even less accessible areas. In this chapter we aim to determine the location and orientation of a ground-level query image by cross-view matching it to a reference database of overhead images. For this task we collect a new dataset with one million pairs of street view and overhead images sampled from eleven U.S. cities; even though this data is not at the scale of entire world, it's big enough to validate our result and can generalize beyond cities. We explore several deep CNN architectures for cross-domain matching – Classification, Hybrid, Siamese, and Triplet networks. Classification and Hybrid architectures are accurate but slow since they allow only partial feature precomputation. We propose a new loss function which significantly improves the accuracy of Siamese and Triplet embedding networks while maintaining their applicability to large-scale retrieval tasks like image geolocalization. This image matching task is challenging not just because of the dramatic viewpoint difference between ground-level and overhead imagery but because the orientation (i.e. azimuth) of the street views is unknown making correspondence even more difficult. We examine several mechanisms to match in spite of this – training for rotation invariance, sampling possible rotations at query time, and explicitly predicting relative rotation of ground and overhead images with our deep networks. It turns out that explicit orientation supervision *also* improves location prediction accuracy. Our best performing architectures are roughly 2.5 times as accurate as the commonly used Siamese network baseline.

3.1 Introduction

In this work we propose deep learning approaches to the problem of ground to overhead image matching. Such approaches enable large scale image geolocalization techniques to use widely-available overhead/satellite imagery to estimate the location of ground level photos. This is in contrast to typical image geolocalization which relies on matching "ground-to-ground" using a reference database of geotagged photographs. It is comparatively easy (for humans and machines) to determine if two ground level photographs depict the same location, but the world is very non-uniformly sampled by tourists and street-view vehicles. On the other hand, overhead imagery densely covers the Earth thanks to satellites and other aerial surveys. Because of this widespread coverage, matching ground-level photos to overhead imagery has become an attractive geolocalization approach [32]. However, it is a very challenging task (even for humans) because of the huge viewpoint variation and often lighting and seasonal variations, too. In this work we try to learn how to match urban and suburban images from street-view to overhead-view imagery at fine-scale. As shown in Figure 3.1, once the matching is done, the results can be ranked to generate a location estimate for a ground-level query.

To address cross-view geolocalization, the community has recently found deep learning techniques to outperform hand-crafted features [14, 15]. These approaches adopt architectures from the similar task of face verification [41, 12]. The method is as follows: a CNN, more specifically a Siamese architecture network [42, 41], is used to learn a common low dimensional feature representation for both ground level and aerial image, where they can be compared to determine a matching score. While being superior to non-deep approaches (or pre-trained deep features), we show there is significant room for improvement.

To that end we study different deep learning approaches for matching/verification and ranking/retrieval tasks. We develop better loss functions using the novel distance based logistic (DBL) layer. To further improve the performance, we show that good representations

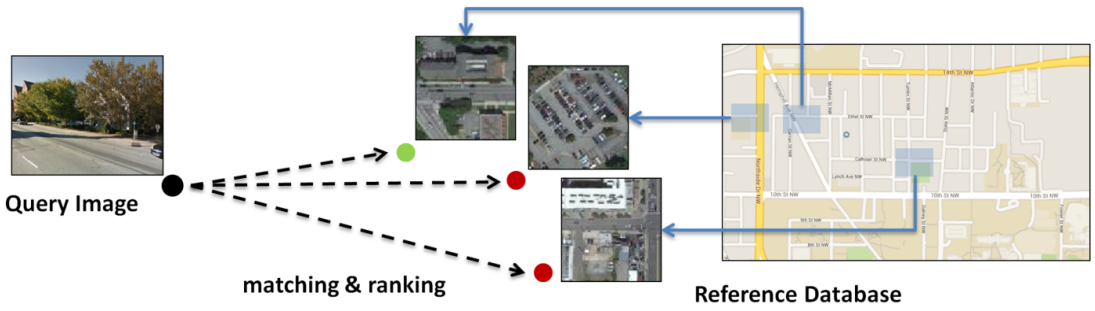

Figure 3.1: Street-view to overhead-view image matching

can be learned by incorporating rotational invariance (RI) and orientation regression (OR) during training. Experiments are performed on a new large scale dataset which will be published to encourage future research. We believe the findings here generalize to similar matching and ranking problems.

3.1.1 Related work

Image geolocalization uses recognition techniques from computer vision to estimate the location (at city, region, or global scale) of ordinary ground level photographs. Early work by Hays and Efros [**hays2008im2gps**] studied the feasibility of this task by leveraging millions GPS-tagged images from the Internet. In [30], image localization is done efficiently by building a dataset of Google street-view images from which SIFT features are extracted, indexed and used for localization of a query image by voting. Lin et al. [32] propose the first ground-to-overhead geolocalization method. No attempt is made to learn a common feature space or match directly across views. Instead, ground-to-ground matching to a reference database of ground-overhead view pairs is used to predict the overhead features of a ground-level query. Bansal et al. [34] match street-level images to aerial images by proposing a feature which encodes facade structure self-similarity. Shan et al. [35] propose a fully automated system that registers ground-based multi-view Stereo models to aerial imagery using traditional multi-view and Structure from Motion technique.

Deep learning has been successfully applied to a wide range of computer vision tasks

such as recognition of objects [5], places [6], faces [12]. Most recently, "PlaNet" [43] made use of a large amount of geo-tagged images, quantized the gps-coordinate into a number of regions and trained a CNN to classify an image's location into one of those regions. More relevant to this work is deep learning applications in cross-view images matching [14, 15]. The most similar published work to ours is Lin et al. [14] which uses a Siamese network to learn a common deep representation of street-view images and 45 degree aerial or bird's eye images. This representation is shown to be better than hand-crafted or off-the-shelf CNN features for matching buildings' facades from different angles. In [15], Workman et al. show that by learning different CNNs for different scales (i.e. using aerial images at certain scales), geolocalization can be done at the local or continental level. Interestingly, they also showed that by fixing the representation of ground-level image, which is 205 categories scores learned from the Places database [6], the CNN will learn the same category scores for aerial images. Most recently, Altwaijry et al. [31] use a deep attentive architecture to match aerial images across wide baselines.

3.2 Dataset of street view and overhead image pairs

We study the problem of matching street-view image to overhead images for the application of image geolocalization. To that end, we collect a large scale dataset of street-view and overhead images. More specifically, we randomly queried street-view panorama images from Google Map of the US. For each panorama, we randomly made several crops and for each crop we queried Google Map for the overhead image at the finest scale, resulting in an aligned pair of street-view and overhead images. Note that we want to localize the *scene* depicted in the image and not necessarily the camera. This is possible since Google panorama images come with geo-tags and depth estimates. We performed this data collection procedure on 11 different cities in the US and produced more than 1 million pairs of images. Some example matches in Miami are shown in Figure 3.2. We make this dataset available to the public.

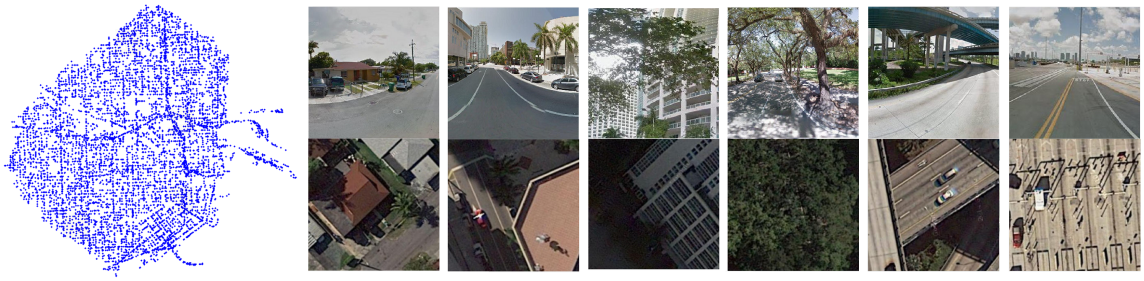

Figure 3.2: On the left: visualization of the positions of all Miami's panorama images that we randomly collect for further processing. On the right: examples of produced street-view and overhead pairs.

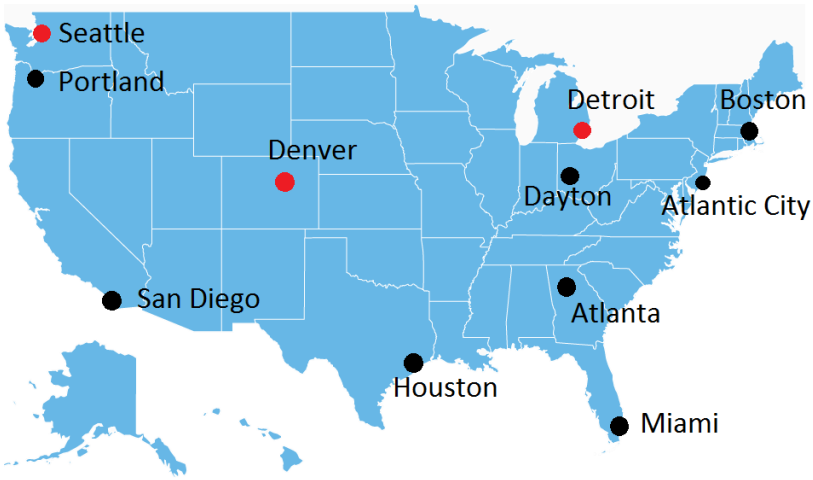

Figure 3.3: Location of cities we chose to build our dataset. Black: we use for training, and Red: we use for testing in our experiments.

Figure 3.3 shows the cities from which we collected data (initially we wanted to use both big and small city/town, but the image quality seems to be quite inconsistent). Majority of the images in the dataset are of rural-like scene because the urban area is relatively small even in big cities.

Some similar attempts to collect a dataset for cross-view images matching task are [14, 15], but neither are publicly available. We expect that the result and analysis here can be easily generalized across other datasets (or other applications like recognizing face or object instead of scene). While the technical aspects are similar, there will be qualitative differences: when training on [14], the network learns to match the facade which is visible

from both views. On [15], the network learns to match similar categories of scenes or land cover types. And on our dataset, the network learns to recognize different fine-grained street scenes.

3.3 Cross-view matching and ranking with CNN

Before considering the ranking/retrieval task, we start with the matching/verification task formalized as following: during training phase, matched pairs of street-view and overhead images are provided as positive examples (negative examples can be easily generated by pairing up non-matched images) to learn a model. During testing, given a pair of images, the learned model is applied to classify if the pair is a match or not.

We use deep CNNs which have been shown to perform better than traditional hand-crafted features, especially for problems with significant training data available. We study 2 categories of CNNs (Figure 3.4): the classification network for recognizing matches and the representation learning networks for embedding cross-view images into the same feature space. Note that the first category is not practical for the large-scale retrieval application and is used as a loose upper bound for comparison.

The second category includes the popular Siamese-like network and the triplet network. We introduce another version of Siamese and triplet networks that use the *distance based logistic* layer, a novel loss function. For completeness we also include the Siamese-classification hybrid network (which will belong to the first category). In this section we will experiment with 6 networks in total.

3.3.1 Classification CNN for image matching

Since our task is basically classification, the first network we experiment with is AlexNet[5], originally demonstrated for object classification (Figure 3.4(a)). It has 5 convolutional layers, followed by 3 fully-connected layers and a soft-max layer for classification. We make several modifications: (1) the input will be a 6-channel image, a concatenation of a street-

Figure 3.4: Different CNN architectures: on the left is the first category: the classification network and the Siamese-classification hybrid network, on the right is the second category: the Siamese network and the triplet network

view image and an overhead image, while the original AlexNet only takes 1-image input , (2) we double the number of filters in the first convolutional layer, (3) we remove the division of filters into 2 groups (this was done originally because of GPU memory limitation) and (4) the softmax layer produces 2 outputs instead of 1000 because our task is binary classification. Similar architectures have been used for comparing image patches [44].

Training the CNN is done by minimizing this loss function:

$$L(A, B, l) = LogLossSoftMax(f(I), l) \tag{3.1}$$

where A and B are the 2 input images, $l \in \{0, 1\}$ is the label indicating if it's a match, $I = concatenation(A, B)$ and $f(.)$ is the AlexNet that outputs class scores.

3.3.2 Siamese-like CNN for learning image features

The Siamese-like network, shown in Figure 3.4(b), has been used for cross-view image matching [14, 15] and retrieval [45, 9]. It consists of 2 separate CNNs. Each subnetwork takes 1 image as input and output a feature vector. Formally, given 2 images A and B, we can apply the learned network to produce the representation f(A) and f(B) that can be used for matching. This is done by computing the distance between these 2 vectors and classifying it as a match if the distance is small enough. During training, the contrastive

loss is used:

$$L(A, B, l) = l * D + (1 - l) * max(0, m - D) \tag{3.2}$$

where D is the squared distance between f(A) and f(B), and m is the margin parameter that omits the penalization if the distance of non-matched pair is big enough. This loss function encourages the two features to be similar if the images are a match and separates them otherwise; this is visualized in Figure 3.5(left).

In the original Siamese network [21], the subnetworks (f(A) and f(B)) have the same architecture and share weights. In our implementation, each subnetwork will be an AlexNet without weight sharing since the images are of different domains: one is street view and the other is overhead.

3.3.3 Siamese-classification hybrid network

The hybrid network is similar to the Siamese in that the input images are processed independently to produce output features and it is similar to the classification network that the features are concatenated to jointly infer the matching probability (Figure 3.4(c)). Similar architectures have been used for used for cross-view matching and feature learning [44, 46, 47, 31].

Formally let AlexNet (f) is consist of 2 parts: the set of convolutional layers (f_{conv}) and the set of fully-connected layers (f_{fc}), the loss function is:

$$L(A, B, l) = LogLossSoftMax(f_{fc}(I_{conv}), l) \tag{3.3}$$

Where $I_{conv} = concatenation(f_{conv}(A), f_{conv}(B))$. We expect this network to approach the accuracy of the classification network, while being slightly more efficient because intermediate features only need to be computed once per image.

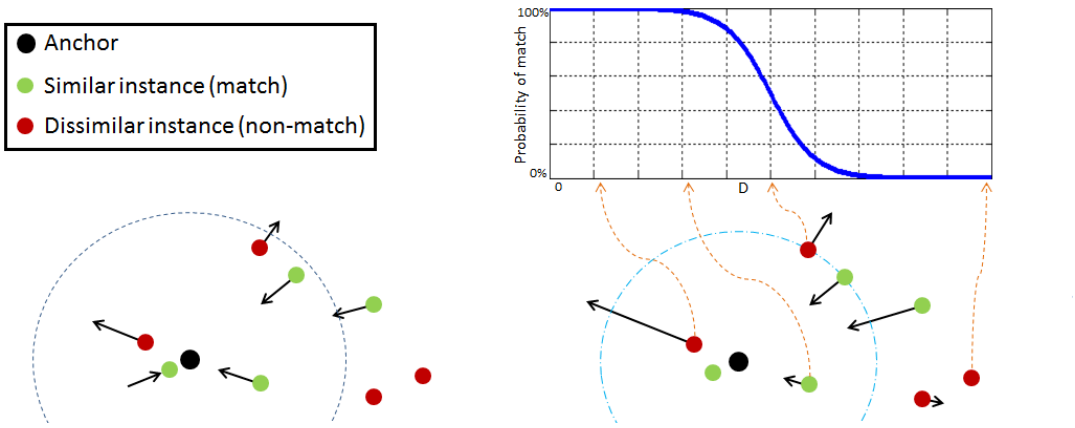

Figure 3.5: Visualization of Siamese network training. We represent other instances (matches and non-matches) relative to a fixed instance (called the anchor). Left: with contrastive loss, matched instances keep being pulled closer, while non-matches are pushed away until they are out of the margin boundary, Right: log-loss with DBL: matched/nonmatched instances are pushed away from the "boundary" in the inward/outward direction.

3.3.4 Triplet network for learning image features

The fourth network that we call the triplet network or ranking network, shown in Figure 3.4(c), is popular for image feature learning and retrieval [48, 24, 49, 50, 13, 51], though its effectiveness has not been explored in cross-view image matching. More specifically it aims to learn a representation for ranking relevance between images. It consists of 3 separate CNNs instead of 2 in the Siamese network. Formally, the network takes 3 images A, B and C as inputs, where (A,B) is a match and (A,C) is not, and minimizes this hinge loss for triplet (which has been explored before its application in deep learning [22, 23]):

$$L(A, B, C) = max(0, m + D(A, B) - D(A, C)) \tag{3.4}$$

Where D is the squared distances between the features f(A), f(B), f(C), and m is the margin parameter to omit the penalization if the gap between 2 distances is big enough. This loss layer encourages the distance of the more relevant pair to be smaller than the less relevant pair (Figure 3.6(left)).

In the context of image matching, a pair of matched images (as the anchor and the

match), plus a random image (as the non-match) is used as training example. With the learned representation, matching can be done by thresholding just like the Siamese network case.

3.3.5 Learning image representations with distance-based logistic loss

Despite being intuitive to understand, common loss functions based on euclidean distance might not be optimal for recognition. We instead advocate loss functions similar to the standard softmax, log-loss.

For the Siamese network, instead of the contrastive loss, we define the distance based logistic (DBL) layer for pairs of inputs as:

$$p(A, B) = \frac{1 + exp(-m)}{1 + exp(D - m)} \tag{3.5}$$

This outputs a value between 0 and 1, as the probability of the match given the squared distance. Then we can use the log-loss like the classification case for optimization:

$$L(A, B, l) = LogLoss(p(A, B), l) \tag{3.6}$$

The behavior of this loss is visualized in Figure 3.5(right). Notice the difference from the traditional contrastive loss.

For the triplet network, we define the DBL for triple as following:

$$p(A, B, C) = \frac{1}{1 + exp(D(A, B) - D(A, C))} \tag{3.7}$$

This represents the probability that it's a valid triple: B is more relevant to A than C is to A (note that $p(A, B, C) + p(A, C, B) = 1$). Similarly the log-loss function is used, so:

$$L(A, B, C) = log(1 + exp(D(A, B) - D(A, C))) \tag{3.8}$$

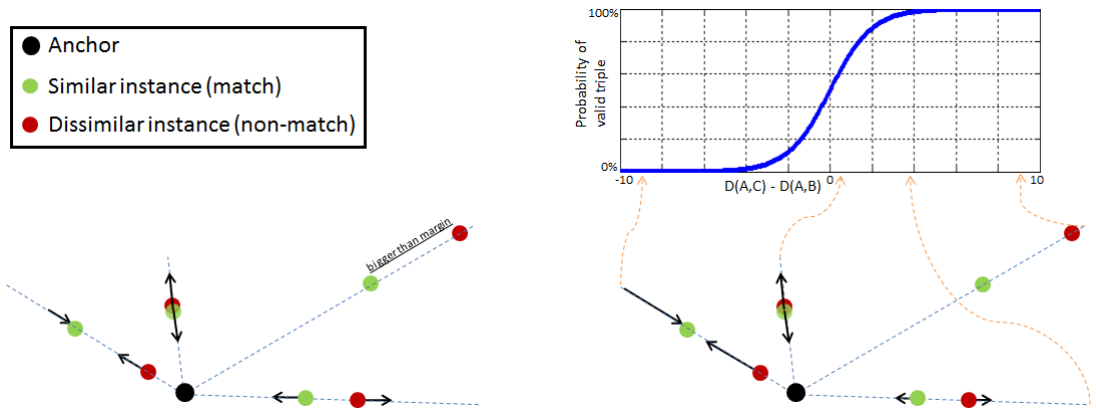

Figure 3.6: Visualization of triplet network training. Each straight line originating from the anchor represents a triple. Left: with triplet/ranking loss, instances are pulled and pushed until the difference between the match distance and the non-match distance is bigger than the threshold, Right: log loss with DBL for triple. Similar to the ranking loss, but instead of relying on the threshold, the "force" depends on the current performance and confidence of the network.

The behavior of this loss is visualized in Figure 3.6(right).

With this novel layer, we obtain Siamese and triplet DBL-Net that allow us to optimize for the recognition accuracy more directly. As with the original loss functions, the learned feature representation can be used for efficient matching and ranking at test time (when the DBL layer is not involved).

Implementation detail: we use m=10; and D(.) is squared Euclidean distance. We do not do feature normalization (L2) in all of our experiment; hence the network can change the scale of the feature and the formulas here can be applied directly. However if there's normalization which basically predefined the scale of the output feature (and therefore the distance between them), it's best to scale the feature by a suitable constant (for example 3) before applying the DBL-log loss. Or equivalently change/validate the steepness and the midpoint of the logistic curve (instead of using the standard logistic function form).

3.4 Learning to perform rotation invariant matching

As we are considering the task of fine-grained street view to overhead view matching, not only spatial but also orientation alignment is important, i.e. rotating the overhead image

according to the street-view's orientation instead of keeping the overhead image north oriented.

We aim to learn a rotation invariant (RI) representation of the overhead images. Similarly, Ke at al [52] studied the problem of shape recognition without explicit alignment. In [53], nearby filters are untied to potentially allow pooling on output of different filters. This helps to learn complex representation without big filters or increasing the number of filters; however that doesn't result in an explicit RI property like we desire. Deep symmetry network [54] is capable of encoding such a property, though its advantage is not significant when training data is sufficient for traditional CNN to learn that on its own. More relevant, [55] uses data augmentation and concatenation of features from different viewpoints. However our training data comes with orientation aligned images (though not the test sets), which can potentially provide stronger supervision during training. In this section we explore techniques to take advantage of such information.

3.4.1 Partial rotation invariance by data augmentation

Training with multiple rotation samples: Rotation invariance (RI) can be encouraged simply by performing random rotation of overhead training images. Although invariance can help to a certain extent, there is a trade-off with discriminative ability. We propose to control the amount of rotation that the matching process will be invariant to, i.e. partial RI. Specifically this is done by adding a random amount of rotation within a certain range to the aligned overhead images. For example a 90° RI is achieved by rotating by an amount from -45° to 45°; 360° RI means fully RI.

Testing with multiple rotation samples/crops: since we don't know the correct orientation alignment at test time, if our representation is only partially rotation invariant, we have to test with multiple rotated version of the original image to find the best one. For example: with 360° RI representation, 1 sample is enough, with 180° RI representation, at least 2 rotation samples (that are 180° apart) are needed. Similar to multi-crop in classi-

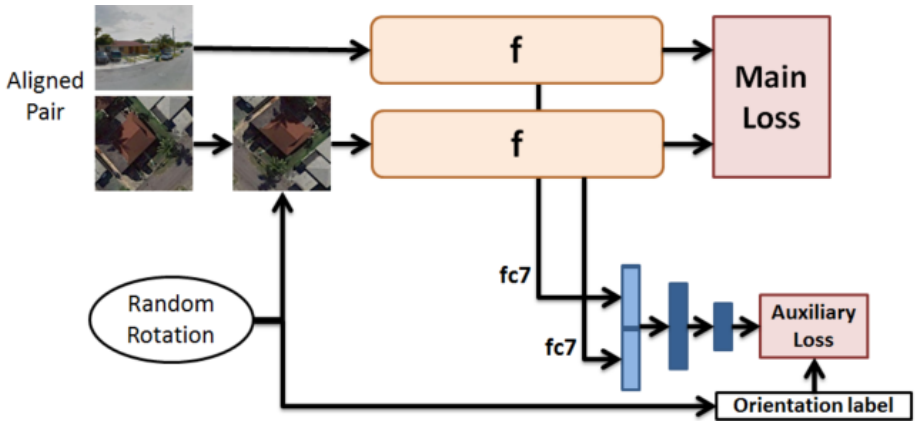

Figure 3.7: Network architecture with data augmentation by random rotation and an additional branch that performs orientation regression

fication tasks, we find that using more test time samples improves the result slightly (e.g. using 16 rotation samples at test time even if the network was trained to be 90° RI).

Multi-orientation feature averaging: as we use more rotation samples than needed, not only one but multiple of them should be good matches. For example testing with 16 rotation, we expect 16 of the them are good matches under 360° RI range, 4 under 90° RI range, etc. Therefore it makes sense to, instead of matching with a single best rotation (nearest neighbor), match with the best sequence of rotations. We propose to, depending on the degree of RI, average the features of multiple rotation samples during indexing time to obtain more stable features. This technique is especially useful in full RI case: all samples are averaged to produce a single feature, so the cost during query time is the same as using 1 sample.

3.4.2 Learning better representations with orientation regression

Next we propose to add an auxiliary loss function for orientation regression, where the amount of added rotation during training can be used as label for supervision. As shown in Figure 3.7, the features from the last hidden layer (fc7) are concatenated, then we add 2 fully connected layers (one acting as hidden layer and one as output layer) and use Euclidean distance as our loss function for regression.

It is known that additional or 'auxiliary' losses can be very useful. For example, ranking can be improved by adding a classification layer predicting category [9, 51] or attributes [10]. In [56], co-training of verification and classification is done to obtain a good representation for faces. Somewhat differently, our auxiliary loss is not directly related to the main task and its label is randomly generated by data augmentation. As the inference is done on 2 images jointly, its effect on each individual's representation can be difficult to interpret. The motivation, beyond being able to predict query orientation, is that this will make the network more orientation-aware and therefore produce a better feature representation for the localization task.

3.5 Experiments

Data preparation: we use our dataset of more than 1 million matched pairs of street-view and overhead-view images randomly collected from Google Maps of 11 different US cities (section 2). We use all the cross-view pairs in 8 cities as training data (a total of 900k examples) and the remaining 3 cities as 3 test sets (around 70k examples per set).

We learn with mini-batch stochastic gradient descent, the standard optimization technique for training deep networks. Our batch size is 128 (64 of which are positive examples while 64 are negative examples). Training starts with a large learning rate (experimentally chosen) and get smaller as the network converges. The number of training iterations is 150k. We use Caffe framework [40].

Data augmentation: we apply random rotation of overhead images during training and use multiple rotation samples during testing (described in Section 4). The effect will be studied in detail in section 5.2. We also apply a small amount of random cropping and random scaling.

Image Ranking and Geolocalization. While we have thus far considered location matching as a binary classification problem, our end goal is to use it for geolocalization. This application can be framed as a ranking or retrieval problem: given a query street view

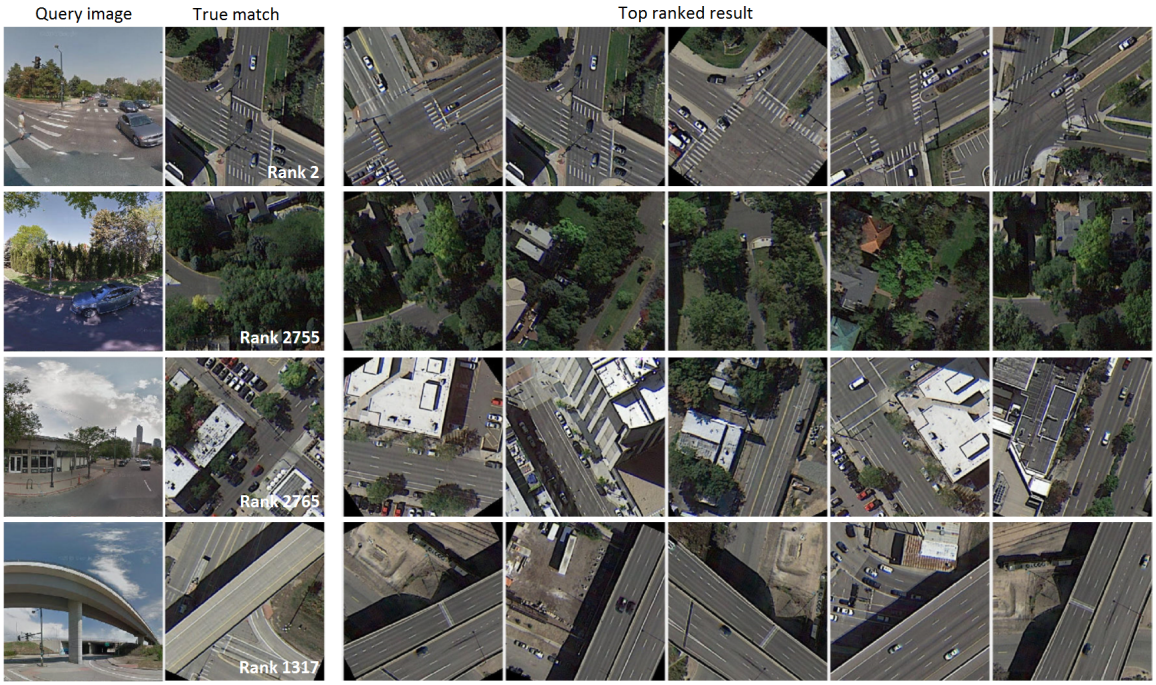

Figure 3.8: Ranking result examples on the Denver test set (reference set of 70k reference images)

image and a repository of overhead images, one of which is the match, we want to rank the overhead images according to their relevance to the query so that the true match image is ranked as high as possible. The ranking task is typically approached as following: the representation learning networks are applied to the query image and the repository's images to obtain their feature vectors. Then these overhead images can be ranked by sorting the distance from their features to the query image's feature. The localization is considered successful if the true match overhead image is ranked within a certain top percentile.

Metrics: we measure both the classification and ranking performance on each test set. The classification accuracy is computed by using the best threshold on the each test set (random chance performance is 50%). We found that this measurement is useful for evaluating classification networks which are hard to apply to ranking on large test sets because of the computational expense of all-pairs comparisons through deep networks. For the ranking task, we use mean recall at top K% as our measurement (the percentage of cases in which the correct overhead match of the query street view image is ranked within

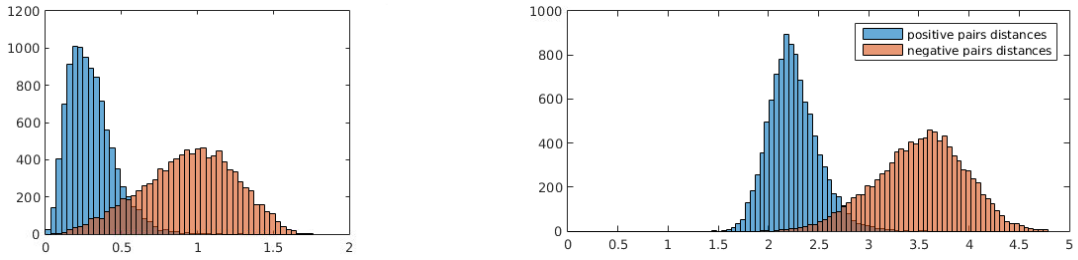

Figure 3.9: Histograms of pairwise distances of features produced by the Siamese network-contrastive loss (left) and the triplet network (right). Note the crowding near zero distance for the Siamese network, which may explain poor performance for fine-grained retrieval tasks when it is important to compare small distances.

top K percentile, chance performance is K%). Some ranking examples are shown in Figure 3.8.

3.5.1 Comparison of CNN architectures

We train and compare 6 variants of CNN described in Section 3. All are initialized from scratch (no pretraining), trained to be 90° RI, and tested with 16 rotation samples. Quantitative comparisons are shown in the top of Table 3.1.

Not surprisingly, both classification networks achieved better accuracy than the representation learning Siamese and triplet networks. This is because they jointly extract and exchange information from both input images. Somewhat unexpectedly, in our experiments the hybrid network is the better of the two. Even-though the 'pure' classification network should be capable of producing the same mapping as the hybrid, it might have trouble learning to process both images from the 1st layer.

Between the Siamese and triplet network, the triplet network outperforms the Siamese by a surprisingly large margin on both tasks. While both networks try to separate matches from non-matches, the contrastive loss function works toward a secondary objective: drive the distance between matched pair as close to 0 as possible (Figure 3.9). Note that this might be a good property for the learned representation to have; but for the task of matching and ranking we found that this might compromise the main objective. One way to alleviate this problem is to add another margin to the contrastive loss function to cut the loss when

Table 3.1: Performance of different networks on different test sets

Task	Classification (accuracy)			Ranking (recall @top 1%)		
Test set	Denver	Detroit	Seattle	Denver	Detroit	Seattle
Section 5.1 experiment (90°RI+16rots)						
Classification network	90.0	87.8	87.7	N/A	N/A	N/A
Classification hybrid	91.5	88.7	89.4	N/A	N/A	N/A
Siamese network	85.6	83.2	82.9	21.6	21.9	17.7
Triplet network	88.8	86.8	86.4	43.2	39.5	35.3
Siamese DBL-Net	90.0	88.0	**88.0**	48.4	45.0	**41.8**
Triplet DBL-Net	**90.2**	**88.4**	87.6	**49.3**	**47.1**	40.0
Section 5.2 (360°RI+OR)						
DBL-Net + 16rots	**91.5**	**90.1**	88.7	**54.8**	**52.7**	**45.5**
DBL-Net + avg16	**91.5**	90.0	**88.8**	54.0	52.2	45.3
Section 5.3						
Triplet eDBL-Net	**91.7**	**89.9**	**89.3**	**59.9**	**57.8**	**51.4**

the distance is small enough [57].

Analysis of Siamese and triplet network's performance has helped us develop the DBL layer. As the result, both DBL-Nets significantly outperform the original networks. While the Siamese with DBL and triplet network with DBL have comparable performances, it seems that the triplet DBL-Net is slightly better at ranking. Note that for most of the experiments we have been conducting, the performance of these two tasks strongly correlate. We use the triplet network with DBL layer for all following experiments.

3.5.2 Rotation invariance

We experiment with partial rotation invariance (RI) and orientation regression (OR) (described in Section 4) for matching and ranking using the triplet DBL-Net. The result is shown in Table 3.2.

As an upper bound, we train a network where overhead images are aligned to the ground truth camera direction of the street view image (1GT). This is not a realistic usage scenario for image geolocalization since camera azimuth would typically be unknown. As expected, the network without RI performs very well when true alignment is provided during testing

Table 3.2: Comparisons of different amount of partial rotation invariance (RI), with and without orientation regression (OR), and different numbers of rotation samples during test time. In this experiment, the triplet network with DBL layer is tested on the Denver test set. 1GT*: in this setting, we test with 1 overhead image aligned using the ground-truth orientation (so the network doesn't have to be RI).

Task	Classification (accuracy)				Ranking (recall @top 1%)			
Number of test rotations	1	4	16	1GT*	1	4	16	1GT*
0° RI (no RI)	63.6	68.5	87.2	95.0	11.0	18.8	37.3	76.2
45° RI	70.9	86.2	89.9	N/A	19.3	36.8	48.1	N/A
90° RI	75.8	89.5	**90.2**	N/A	24.7	44.7	**49.3**	N/A
180° RI	82.7	89.2	89.6	N/A	31.2	43.0	45.6	N/A
360° RI (full RI)	87.7	88.5	88.9	N/A	36.8	40.0	41.9	N/A
90° RI + OR	74.3	88.6	89.4	N/A	23.1	43.4	47.4	N/A
360° RI + OR	90.9	91.3	**91.5**	N/A	50.9	53.2	**54.8**	N/A
360° RI + OR + avg16	**91.5**	N/A	N/A	N/A	54.0	N/A	N/A	N/A

(1GT), but performs poorly otherwise. This baseline shows how challenging the problem has become because of orientation ambiguity. As the degree of RI during training is increased, the performance improves.

Observe that fewer numbers of test time rotated crops/samples doesn't work well if the amount of RI is limited. The full RI setting is the best when testing with a single sample. As the number of rotations increase, the performance improves, especially for the partially RI networks. Using 16 rotations, the 90° RI network has the highest performance. It might be the best setting for compromising between invariance and discriminate power (this might not be the case when using hundreds of samples, but we found that it's not computationally practical and the improvement is not significant).

Orientation regression's impact on the 360° RI network is surprisingly significant; its performance improves by 30% (relatively). However OR doesn't affect 90° RI network positively, suggesting that the 2 techniques might not complement each other. It's interesting that the OR is useful even though its effect during learning is not as intuitive to understand as partial RI. As a by-product, the network can align matches. The orientation prediction has an average error of 17° for the ground truth matching overhead image and is

discussed more in the supplemental document.

Finally we show the effect of applying multi-orientation feature averaging on 360° RI + OR network. By averaging the feature of 16 samples, we obtain comparable performance to exhaustively testing with 16 samples (result on all 3 test sets is shown in the 2nd part of Table 3.1). Though not shown here, applying this strategy to partial RI networks also slightly improve their performances.

3.5.3 Triplet sampling by exhausting mini-batch

To speed up the training of triplet networks with the triplet hinge loss, clever triplet sampling and hard negative mining is usually applied [50, 13, 24]. This is because the triplet not violating the margin does not contribute to the learning. However it can skew the input distribution if not handled carefully (for instance, only mine hardest examples); different schemes were used in [50, 13, 24].

On the other hand, our DBL-log loss is practically a smoothed version of the hinge loss. We propose to use every possible triplet in the mini-batch. We experiment with using a mini-batch of 128 pairs of (matched) images. Since each image in our data has a single unique match only, we can generate a total of 256 * 127 triplets (256 different anchors, 1 match and 127 non-matches per anchor). This is done within our exhausting DBL log loss layer implementation (eDBL); hence the cost of processing the mini-batch is not much more expensive. In a similar spirit, recent work[26] proposes a loss function that considers the relationship between every examples in each training batch.

We train a triplet eDBL-Net+360°RI+OR+avg16. Its effect is very positive: the convergence is much faster, after around 30k iterations the network achieved similar performance as in previous experiments where each network was trained with 150k iterations using the same batch size. After 80k iterations, we achieve even better ranking performance, shown at the bottom of table 3.1.

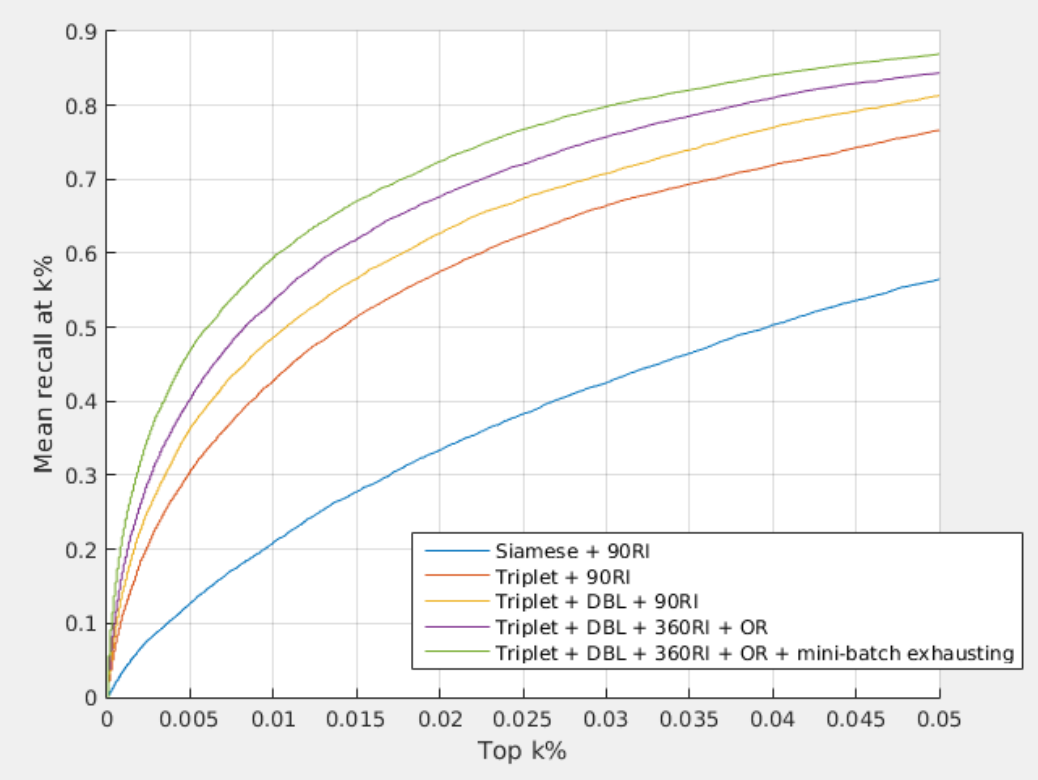

Figure 3.10: Ranking performance on Denver test set

3.5.4 More ranking result

Figure 3.10 shows the ranking performance of some networks that we have described earlier. The Siamese network baseline doesn't perform well relatively suggesting it is not suitable for ranking application. Each of our proposals (DBL + IR + OR + mini-batch exhausting) helps to improve the triplet network significantly.

In figure 3.11 we show some image geolocalization examples. Assuming the position of the overhead images' scenes is known, we can infer the likely position of the scene in streetview image.

3.5.5 Residual network

One can benefit from using deeper network. Here we train a ResNet-101 model [58] and compare it with AlexNet version, the result is shown in table 3.3

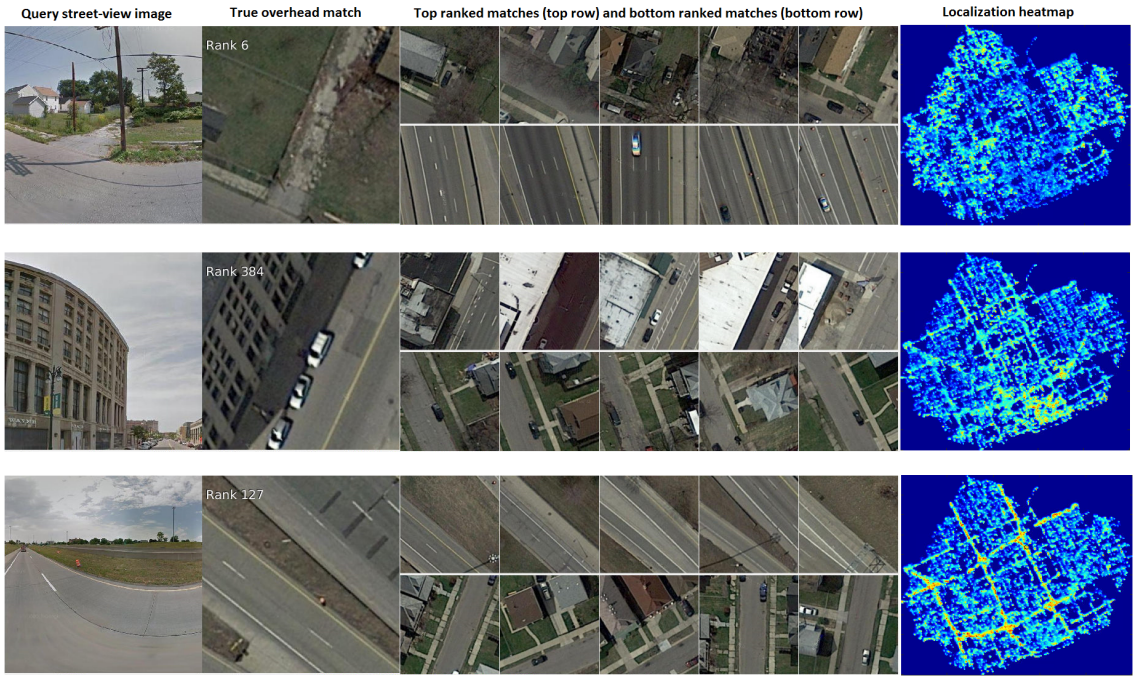

Figure 3.11: 3 geolocalization examples on the Detroit city test set (85,345 overhead-view images)

Table 3.3: Compare AlexNet vs ResNet-101

Task	Classification (accuracy)			Ranking (recall @top 1%)		
Test set	Denver	Detroit	Seattle	Denver	Detroit	Seattle
eDBL-AlexNet	91.7	89.9	89.3	59.9	57.8	51.4
eDBL-ResNet-101	**92.4**	**91.5**	**91.5**	**60.7**	**64.0**	**58.4**

Figure 3.12: conv1 filters learned by the classification network

3.5.6 Network visualization

Conv1: first we visualize the first convolutional layer (named conv1 in AlexNet) learned by these networks. Figure 3.12 shows conv1 of the learned classification AlexNet. Since the input of our (modified) network is a 6-channel image, each convolutional filter also has 6 channels. We split it into 2 parts: channel 1-3 weights (which apply to street-view image) and channel 4-6 weights (which apply to overhead image).

A quick observation is that most the weights are (noisy) zero (gray-color) in either channel 1-3 or 4-6. This indicates that even though this network can combine and exchange information between 2 images, most filters in the first layer only focus on extracting feature from 1 image only. To be sure, we compute the standard deviation of channel 1-3 weights and channel 4-6 weights of each filter and plot all of them in figure 3.12-right. Most filters have 1 std higher than the others and none has both high std value. Another observation is that there's more filters focusing on street-view image than overhead image. In fact, the number of filters having higher channels 1-3 weights std is 111 (out of 192). One explanation is that the scenes in street-view images have greater variation than that of overhead images, hence needing more filters' focus to learn.

In figure 3.13 we show conv1 filters learned by a representation learning networks are similar. Notice the difference between filters of the street-view image and overhead image. These filters are similar to their counterpart in the classification network.

Output feature activation: It's difficult to visualize the features learned in the other layers. In object recognition, they usually detect similar objects or objects' parts [59]. In

Figure 3.13: conv1 filters learned by the representation learning network

cross-view image matching, they detect buildings with similar structural patterns [14]. Our features from the classification network learn to detect similar scenes (or pairs of scenes, in case of classification network). Figure 3.14 shows some images with extreme big value of an output feature in first 5 columns, and images with extreme small value of that same feature in the last 5 columns.

3.5.7 Orientation regression performance

Our network with auxiliary OR loss is capable of predicting the orientation difference between street-view image and overhead-view image; though it's only a by-product and not used for our image geo-localization application. Here we report the network's performance on orientation prediction.

We compute the difference (in degree) between the true orientation and the predicted orientation; the average error (absolute difference) is around $17°$. We plot the histogram of these differences on the Denver test set in figure 3.15. Notice most fall close to $0°$, but there's a very small peak around $-180/180°$. This represents cases in which the scene looks symmetrical from aerial view point. We show some examples prediction in figure 3.16.

3.5.8 Comparison on another cross-view dataset

Most state-of-the-arts for verification or ranking using Siamese or triplet network has been demonstrated to be superior to using shallow or pre-trained features. Our DBL can help to

Figure 3.14: Examples of images with extreme activation value

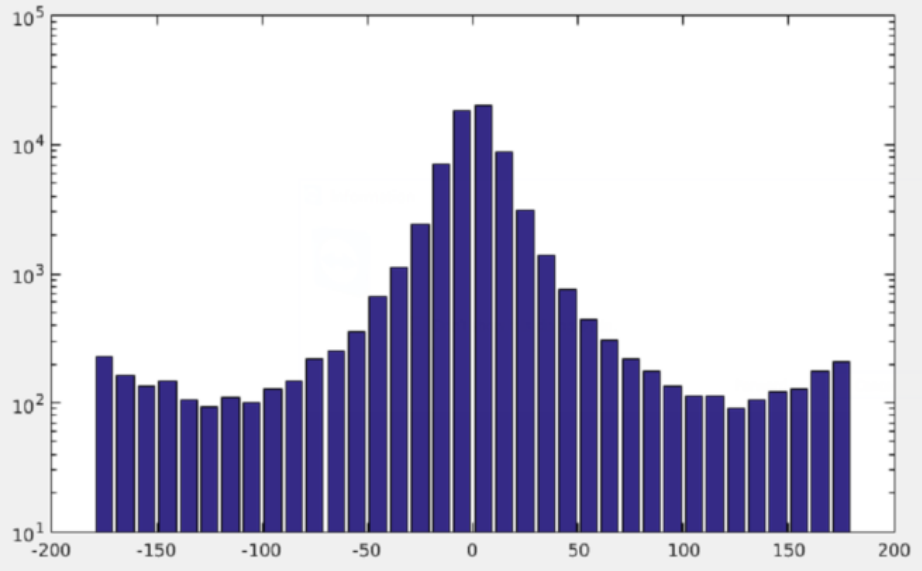

Figure 3.15: Histograms of difference between predicted orientation and true orientation.

Figure 3.16: Orientation prediction examples: first row is the street-view images, second row is the ground-truth aligned overhead images and third row is the alignments using predicted orientation.

further improve the performance.

We experimented on a smaller scale cross-view dataset from [14]. The dataset has around 80k pairs of matched street view images and aerial images in 7 cities; the task is to train on 31k pairs and test the ranking performance on the rest.

We train triplet network and triplet DBL-Net, both initialized from scratch. With mini-batch exhausting, the network fits really fast and begins to overfit after only 5k iterations (batch size: 32 pairs). To deal with that we apply heavy random cropping and random rotation within 10 degree. We run the training for 30k iterations; the result is shown in table 3.4. Triplet eDBL-Net seems to outperform [14] and traditional triplet network on most test sets (though it might not be directly comparable because our training data is slightly smaller than what has been originally used in [14]).

3.6 More feature extraction visualizations

Spatial attention: if we use ResNet-18 as the backbone architecture, we can measure the magnitude of feature vectors at the last convolutional layer. Since the image feature is

Table 3.4: Ranking performance (Recall at 1%) on [14]'s dataset

Test set	SF	Charleston	Chicago	SD	Tokyo	Rome	Lyon
Siamese [14]	22.4	22.6	8.6	23.2	7.3	13.0	**11.7**
Triplet (e-)Net	26.0	33.1	12.3	24.5	7.3	13.6	8.5
Triplet eDBL-Net	**33.8**	**40.8**	**18.2**	**32.0**	**10.2**	**17.2**	11.1

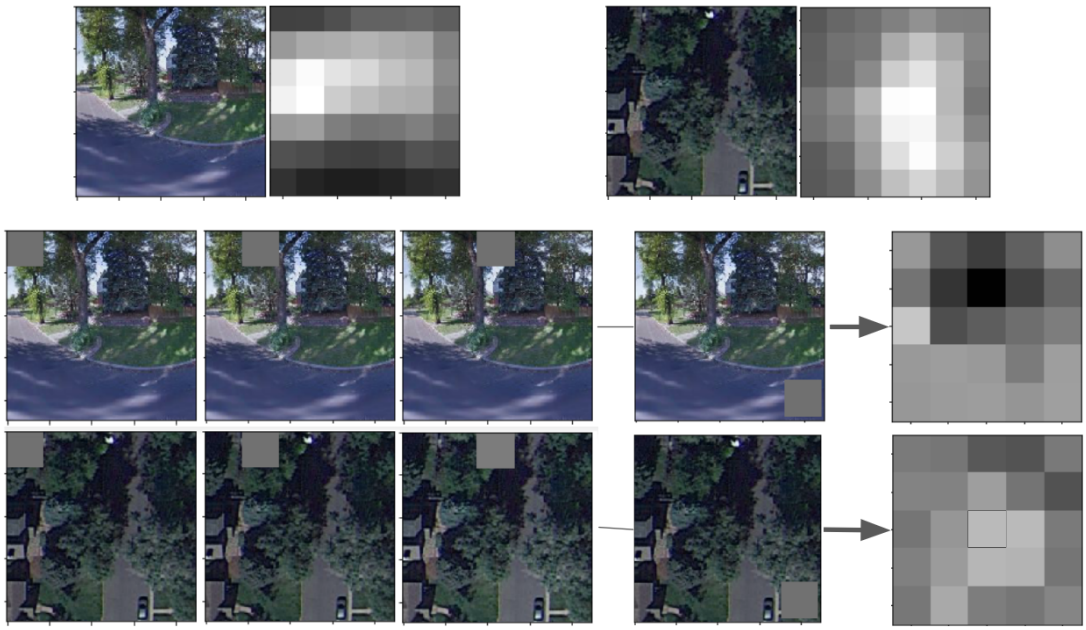

Figure 3.17: Top row: spatial attention; bottom rows: spatial importance.

computed from average pooling of this 2D spatial feature map, the magnitude represents the spatial attention: the contribution at each spatial location to the final image feature. We show one such example in the top row of Figure 3.17; brighter region means bigger feature magnitude (stronger attention).

Spatial importance: another visualization we can produce is a heat map of change in distance between streetview and overhead pair when sliding an occluding window over the image; this has been used in [59, 43]. We call this the spatial importance map. An example is shown in later rows of Figure 3.17; a gray region means no change in the distance (not important), while brighter means bigger distance (caused by removal of important regions) and darker means smaller distance (caused by removal of distracting regions).

More examples of spatial attention and spatial importance are shown in Figure 3.18. It

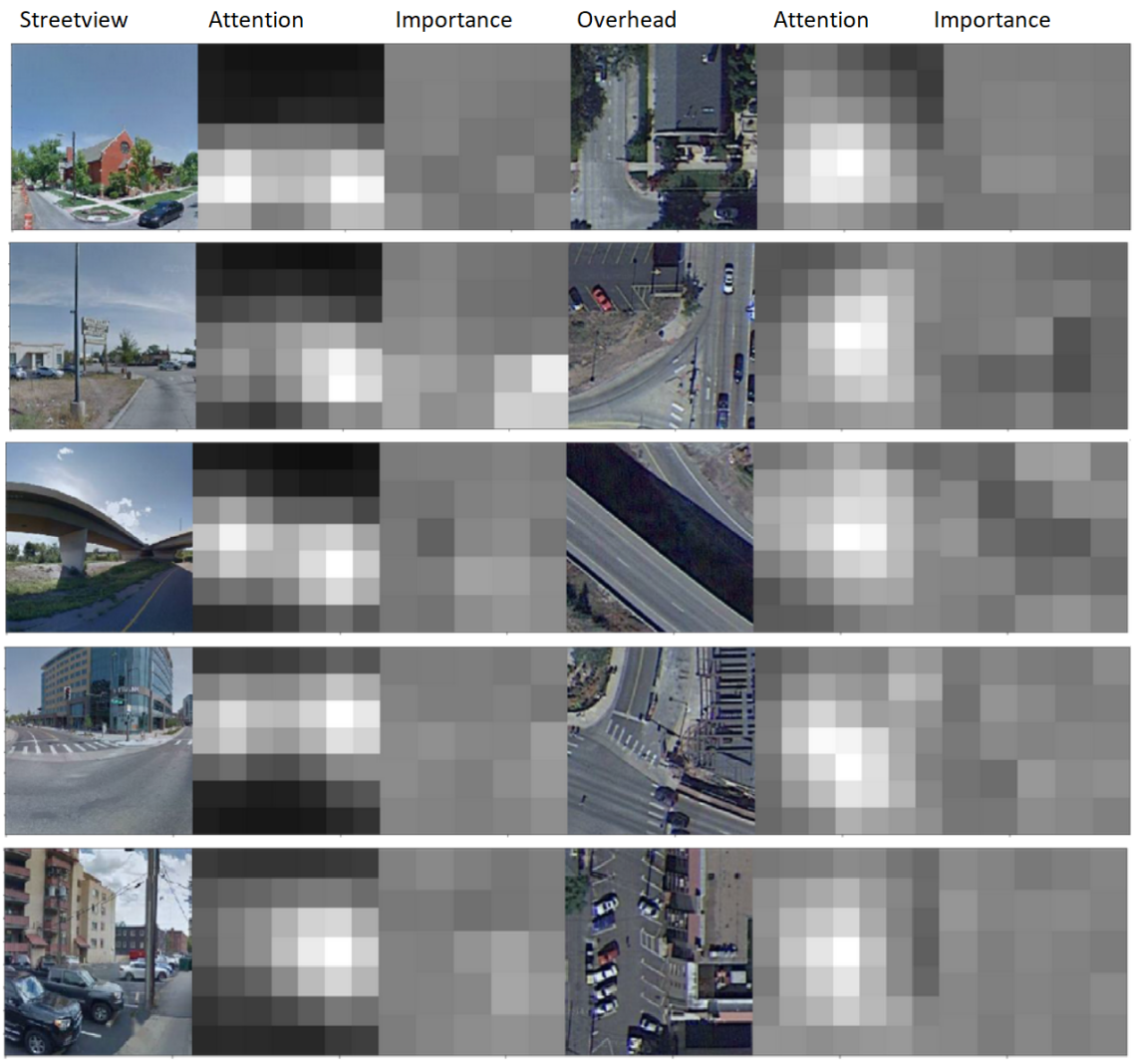

Figure 3.18: Some spatial attention and spatial importance visualization examples.

seems that the streetview attention is stronger on the street and high depth regions, not the ground or sky, in order to capture the scenery. From the overhead viewpoint, the attention is slightly more uniform, often focus in the center and also along the streets. The importance map is somewhat correlated, like regions with weak attentions are usually gray (not important) such as sky in the streetview image.

3.7 Conclusion

We introduce a new large scale cross-view data of street scenes from ground level and overhead. On this dataset, we have experimented with different CNN architectures extensively; the reported results and analysis can be generalized to other ranking and embedding problems. The result indicates that the Siamese network with contrastive loss is the least competitive even though it has been popular for cross-view matching. Our proposed DBL layer has significantly improved representation learning networks. Last but not least, we show how to further improve ranking performance by incorporating supervised alignment information to learn a rotational invariant representation.

CHAPTER 4

GENERALIZATION IN METRIC LEARNING: SHOULD THE EMBEDDING LAYER BE THE EMBEDDING LAYER

Previous chapters explored image geolocalization task with deep learning and has demonstrated that image retrieval approach can be very effective. This chapter focuses on deep metric learning (DML) with fine grained image retrieval application as main testbed (for convenience of analysis and comparison to the literature). Precisely, it studies DML under small to medium scale data as we believe that better generalization could be a contributing factor to the improvement of previous fine-grained image retrieval methods; it should be considered when designing future techniques.

In particular, we investigate using other layers in a deep metric learning system (besides the embedding layer) for feature extraction and analyze how well they perform on training data and generalize to testing data. From this study, we suggest a new regularization practice where one can add or choose a more optimal layer for feature extraction. State-of-the-art performance is demonstrated on 3 fine-grained image retrieval benchmarks: Cars-196, CUB-200-2011, and Stanford Online Product.

We also attempt to apply to scene image retrieval datasets Paris and Oxford, and crossview image retrieval dataset for geolocalization task (chapter 3) and discuss why the proposed practice might not be effective in these cases.

4.1 Introduction

We study small to medium scale deep metric learning (DML) with application to fine-grained image retrieval, from the perspective of generalization. In particular, analyzing training performance could lead to new insight; for example, He at al [60] showed deeper network under-perform on training data and proposed residual connection to overcome that

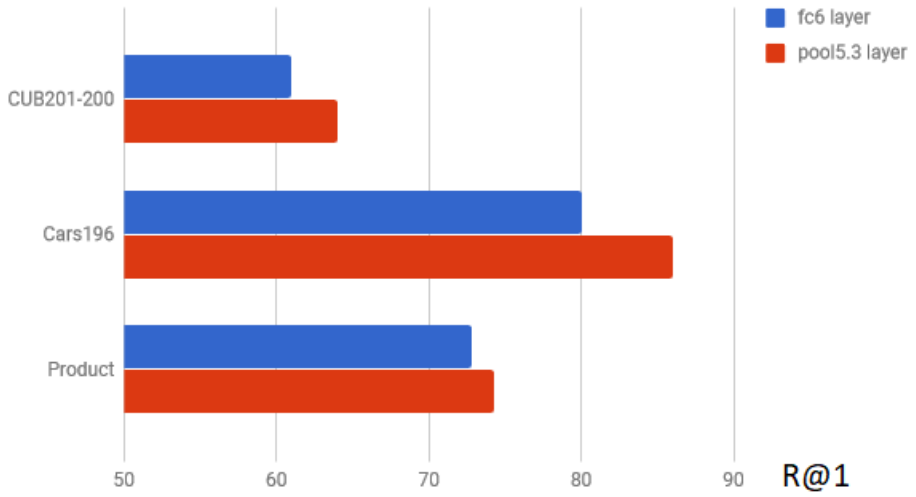

Figure 4.1: Recall at rank 1 performance on 3 benchmarks of the embedding layer (fc6) vs the layer before it (pool5.3)

obstacle.

Deep learning has helped to advance many computer vision tasks, including fine-grained image retrieval: identifying reference images semantically similar to the input/query image. The state of the art approach is to learn an image feature extractor network by using a DML loss function that encourages semantically similar images to have similar features. Hence retrieval can be efficiently done by looking up the nearest neighbors in the feature space.

Given a trained network (for any task, not necessary DML only), the output of any layer can be used as image feature. DML works often use the "embedding" layer: the last one in the network that is the input of the loss function. We train a network using DML and decide to measure the performance of other layers; interestingly the layer before the embedding layer often outperform it, as shown in Figure 4.1. Upon further investigation, it is found that while the last layer performs the best on training data, it doesn't generalize well.

Motivated by above observation, we present here a study of generalization in the context of small to medium scale DML. Niche areas such as fine-grained image retrieval might not have the usual data abundance assumption, so common practice is to finetune a network

that is pretrained on another large-scale dataset. In such scenario, it is very easy to fit the training data and so the system's generalization ability is an important factor.

We explore the following hypotheses:

1. The embedding layer is not pretrained: so it does not generalize as well as the penultimate layer (pool5.3).

2. Pool5.3 layer is shallower: shallow networks fit training data worse but generalize better than deeper networks.

3. Distance from the loss layer: the closer a layer to the loss, the better its feature fits training data and the worse it generalize to test data.

We show, from our analysis, how to train a simple DML pipeline that outperforms state-of-the-art on 3 fine-grained image retrieval benchmarks CUB-200-2011, Cars-196, and Stanford Online Products. Note that we are not proposing a new method; the finding here is potentially applicable to previous works to improve their performance, as shown in our experiments.

4.2 Related Works

Deep metric learning in computer vision has many applications, from generic image search [24] to sketch-based search [11], image geolocalization [16, 61], people re-identification [62, 63] and face recognition [13, 64].

The popular approach is to use a Siamese [65, 41] or triplet network [48, 24, 16]: training examples are put in the form of pairs or triplet, with the label being whether they are similar or not; and either the contrastive loss or triplet hinge loss is used for training, it encourages similar images to have similar features and dissimilar images to have different features. Most recent advances of DML focus on example mining strategy and/or better loss function.

The Multibatch method [66] performs metric learning on all possible pairs from the mini-batch and show that it reduces the variance of the estimator and significantly speed

up the convergence rate. In [16], a similar trick called mini-batch exhausting improves retrieval performance by simply extending the learning to all possible triplets in the mini batch. In [63], it is studied under the name of batch-all and batch-hard. Kihyuk Sohn [67] used a similar batch construction strategy and proposed the multiclass n-pair loss which improves upon the triplet loss.

In [68], Harwood et al proposed a smart mining procedure to select effective samples from the whole training data; this helps the training to converge faster. Huang el al [69] proposed a position-dependent deep metric unit to adaptively select hard examples taking into account the local neighborhood. In [70], Wu et al propose to sample uniformly w.r.t. their relative distance; this distance weighted sampling is shown to be more effective than random or the common semi-hard sampling.

Wang et al [71] proposed a new loss that constraints the angle of the triplet triangle and demonstrate favorable properties over the traditional distance-based hinge loss for triplet. Different from previous works, the proxy-based loss [72] behaves like a classification loss; each training example is compared against learned proxies, not with each other in a pair, triplet or cluster, hence eliminate the need of example mining.

In [73], multiple embeddings are learned at the same time, later one would focus more on examples that are deemed hard to previous ones; the output embedding is a concatenation of all learned embedding. In a similar spirit, Yuan et al [74] propose to learn a set of embeddings organized in a cascaded manner, in which easy training examples are filtered out sequentially and only the hardest one reach the last embedding loss.

4.3 Studying How Well a Layer Generalizes

We'll start with the experiments result and analysis on the small scale dataset Cars196 [75]. Further experiments on other datasets will be discussed later.

The set up will be the image retrieval task; we want to learn a deep network for extracting image features such that images that are semantically similar are close to each other.

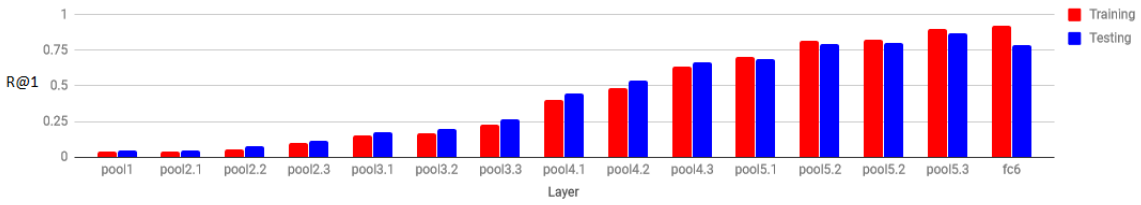

Figure 4.2: R@1 performance of different layers on training and test set of Cars-196.

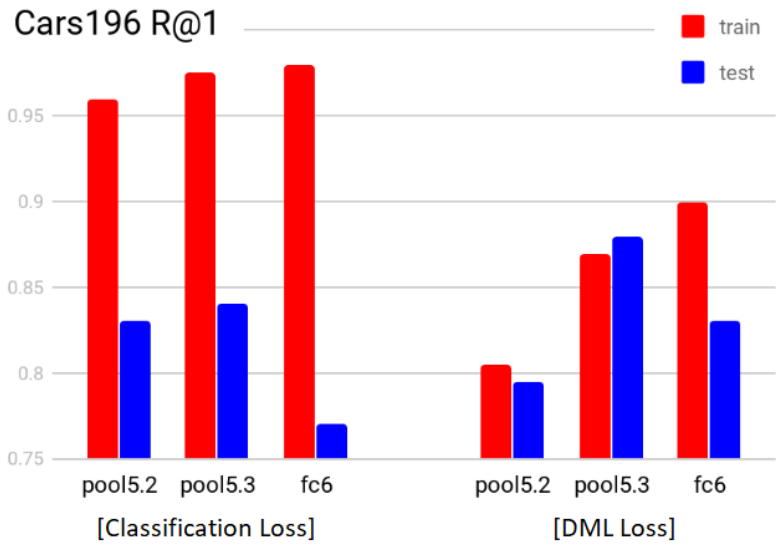

Figure 4.3: R@1 performance of different layers on training and test set of Cars-196. Left: the variant trained with classification loss. Right: our baseline network trained with DML loss.

The common metric Recall at rank 1 (R@1) is used, which is the percentage of test images whose first nearest neighbor (in the feature space) is of the same class.

Applying DML approach, we train the VGG network from an ImageNet-pretrained model. We will describe our implementation in the next section, but note that we are not proposing a new method, the observation here is consistent with different DML systems.

In Figure 4.2, we show the retrieval performance on training data and testing data when using output of different layers as the image feature (we perform max pooling so that all features have similar dimensionality of 512, the raw feature outputs of early layers are too big to be practical and actually perform worse). We observe the performance increases as deeper layers in the network are used, except the last layer doesn't improve upon the layer before it on test data. In Figure 4.3-left, we plot the performance of the last 3 layers (pool5.2, pool5.3, and fc6) changing as the network is trained: although fc6 improves and surpass pool5.3 on training data as expected, it doesn't generalize as well, eventually resulting in worse test time performance. This goes back to the fact that small to medium scale training is more susceptible to over-fitting than under-fitting.

4.3.1 The role of better loss and training strategy

Our system has no problem over-fitting such small-scale training data if trained long enough (for example over 90% R@1 can be achieved in 100 epochs). We speculate that it is similarly easy for other systems to overfit. So a contributing factor to previous improvements could be better generalization, which can be affected by the loss function, training and architecture design or regularization techniques.

We provide an example: we train the same baseline network, but replace the DML loss with classification-output-fc7 and classification-loss (Softmax-CrossEntropy); the result is shown in Figure 4.3-right. Under this loss, the fitting and generalizing behavior is quite different: all 3 layers fit better, but generalization is worse.

Note that classification loss has been previously shown to help improving retrieval per-

formance in multi-task/co-training, especially in large-scale training setting where fitting data really well is the more important factor [64, 76]. Even though the experiment result is different from ours, Horiguchi et al [77] argue that classification loss is advantageous when the size of training data (samples per class) is large. In [61], it is argued that learning with classification is more effective than metric learning because training data is very diverse/noisy.

4.3.2 Is pretrained layer better for embedding

Now we know that pool5.3 can generalize better than fc6; the first possible explanation for that is: conv5.3 (and all layers before it) is pretrained on ImageNet while the embedding layer fc6 is added later for this task and initialized from scratch. We compare 2 cases in Figure 4.4: NetA, the same baseline we are using, and NetB, where both conv5.3 and fc6 are initialized from scratch; for reference, we also include the case of NetC where the whole network is initialized from scratch. It can be seen that with one or more layers initialized from scratch, the performance of most degrades accordingly. Except for fc6, directly under loss layer, can continue fit the training data better if trained long enough.

The training performance of fc6 is better than pool5.3 which in turn is better than pool 5.2. This is expected, as the feature obtained from a layer that is closer to the loss layer work better on training data.

Performance on test data is more interesting:

- NetB:pool5.2 is much better than NetA:pool5.2. It could be conv5.3 being initialized from scratch has a regularization effect. So pretraining a layer might be good for layers after it but has a ”bad effect” for layer before it?

- Pool5.3 performs the best, whether it is pretrained or not. In fact NetB:pool5.3 outperforms both NetB:fc6 and NetA:fc6. It is also better than the NetB:pool5.2 (where conv5.2 is pretrained).

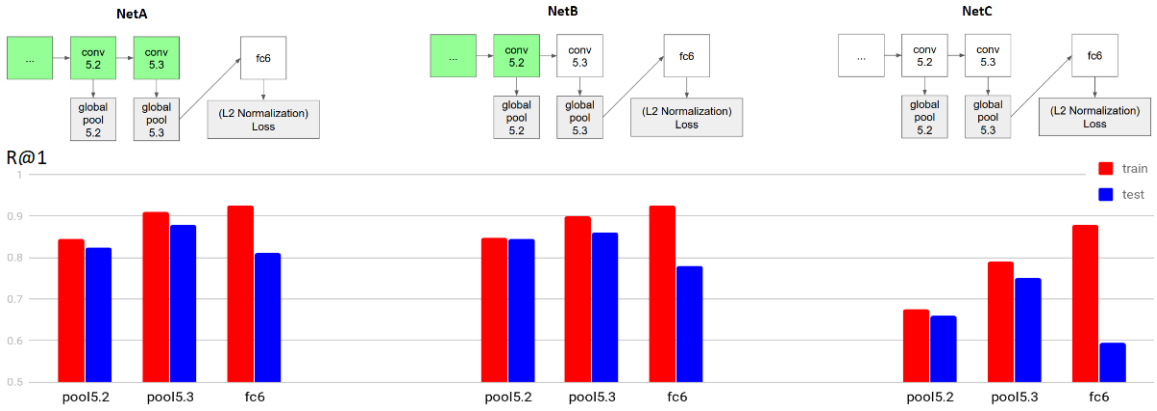

Figure 4.4: R@1 performance of different layers on training and test set of Cars-196. Green box: pretrained layer, white box: initialized from scratch layer, gray box: parameterless layer. Best viewed in color.

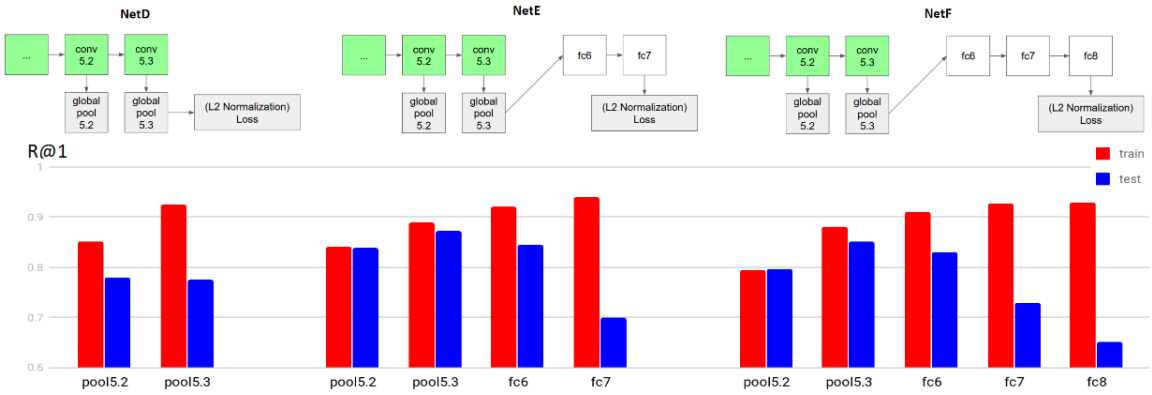

Figure 4.5: R@1 performance of different layers on training and test set of Cars-196. Compared to NetA in Figure 4, the position of the loss function is changed. Best viewed in color.

So while conv5.3 being pretrained does indeed help improve performance, this alone does not explain why pool5.3 is better than fc6.

4.3.3 The depth and the loss layer's position

We explore other potential explanation: layer's depth and the position of loss function. We move the loss layer up/down and introduce 3 new cases, shown in Figure 4.5.

- NetD: similar to NetA except we remove fc6, so the loss is now directly after pool5.3.
- NetE: similar to NetA except we add another fully connected layer fc7.
- NetF: similar to NetA except we add 2 fully connected layer fc7 and fc8.

All new layers have the same output size 512 with ReLU activation between them (not shown in the figure).

We look at the test performance of a layer when it is directly before the loss layer and when it is 2 layers away from the loss layer: pool5.3 in NetD is significantly worse than NetA (even though it's slightly better on training data). Interestingly, we observed similar result for fc6 layer (NetA vs NetE) although the difference is much smaller; and for the case of fc7 (NetE vs NetF) the difference is not significant.

The last important observation is that: when initialized from scratch, and not directly under the loss layer, pool5.3 (NetB) and fc6 (NetE) performance are actually not too different. When directly under the loss layer, pool5.3 (NetD) can be slightly worse than fc6 (NetA).

The position of the loss layer w.r.t. the feature extraction layer seems to play an important role in explaining why one layer is better than the other. Hence to get the best possible test time performance, one should consider using other layers beside the last layer; another option is to distance the intended output layer from the loss layer so that it has a less direct influence, therefore reduce overfitting.

As a best practice, we suggest using NetA:fc6 and NetA:pool5.3 respectively in the cases of less over-fitting to more over-fitting. In small-scale setting, NetE:pool5.3 can also do well; and NetE:fc6 can be used when one can not control the pretraining step but want to define the embedding dimension. Note that when considering these options, it should be taken into account that their behavior can be different with not only different dataset/task, but also different loss function and training strategy.

4.4 Implementation Detail

Our thesis is that generalization is an important factor. With that in mind, we describe the DML design that we choose to work with. Note that we are not proposing a new method, all components here already exist or can be easily implemented; though we will release the source code and models from this study.

4.4.1 Base architecture

We use the VGG-16 architecture with conv layers only (fc6, fc7 and fc8 are removed), followed by a global pooling layer. This architecture is popular for scene image retrieval task [18, 17, 19, 20]. Although more sophisticated pooling method can be better (NetVLAD [18], R-MAC [17] or GeM [78]), we use the simple MAX pooling.

We employ BatchNorm [79] in our system, which is known to make training efficient and also provide regularization effect. Greff et al [80] shows, with ResNet and Highway network making optimization easier, the role of BatchNorm becomes purely regularization (higher training loss, lower testing error). Dropout is also experimented with but not as effective as a regularizer.

We use PyTorch as the framework for experimenting, which comes with VGG-BatchNorm pretrained on ImageNet.

4.4.2 Feature extraction layers

A new fully connected layer fc6 is added at the end of the network, this is the traditional output layer or embedding layer. In this work we experiment with using, not only this last layer, but also other layers before it as the feature extraction layer.

In the experiment from previous section, we also have other variants (NetE and NetF) where we further add fc7 and fc8. All of them have the same output size of 512.

4.4.3 Training mini-batch construction

While DML loss often involves looking at pair or triplet as the example, recent works construct a batch of images as input instead, and only form pairs/triplets/clusters at the loss layer. Hence the mini-batch can be constructed randomly in any way as long as a lot of similar/dissimilar pairs/triplet can be formed (for example [16, 67]).

In case the data comes with the class label, we can randomly sample p classes ($p > 1$), k images per class ($k > 1$) resulting in a batch size of $m = pk$. Under fixed m, maximizing p will increase the diversity leading to more stable gradient; even though the number of possible triplets, $pk(k-1)(pk-k)$, is reduced.

4.4.4 Loss function and example mining

Aside from [19] and [68] that actually perform hard example mining on the whole training data, most mining strategies operate within mini-batch, which is constructed randomly uniform. Hence all these approaches can simply be formulated as a loss function operating at mini-batch level looking at all possible pairs/triplets/clusters and weight them differently. For instance, the recently proposed focal loss for classification task [81] quickly diminish the contribution of easy examples.

Hence we are not doing any explicit hard example mining. Given the batch b of output image features, our loss is defined as:

$$L(b) = \frac{1}{M} \sum_{triplet(a,p,n) \subset b} tripletloss(a, p, n)$$

where $M = \sum_{triplet(a,p,n) \subset b} 1$, the total number of valid triplets in a batch; (a, p, n) is a triplet of the anchor, the similar and the dissimilar instance respectively. We follow [16, 67, 63] and use the smooth loss function (which shown to be better than the traditional hinge loss):

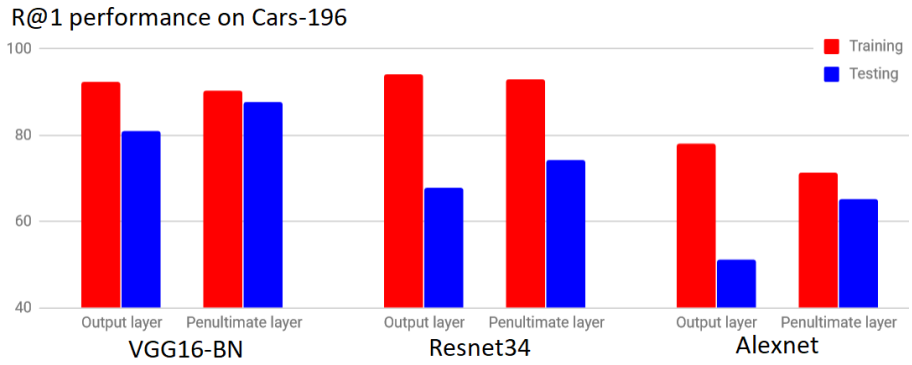

Figure 4.6: Comparing the performance of output layer vs the layer before it when using 3 different backbone architectures.

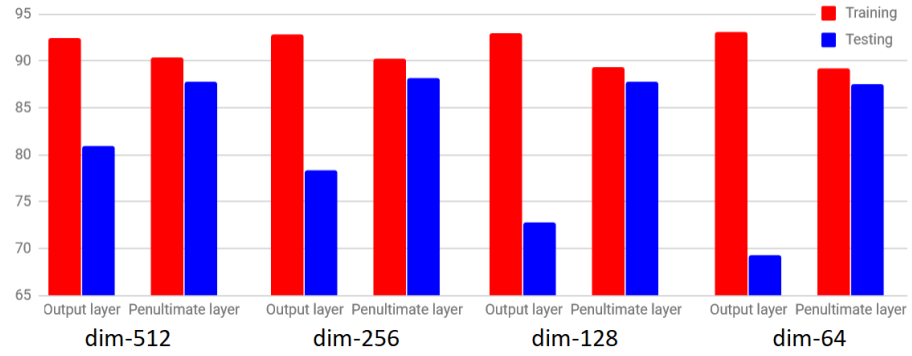

Figure 4.7: Comparing the performance of output layer vs the layer when varying the output layer's dimensionality.

$$tripletloss(a, p, n) = log(1 + exp(d(a, n) - d(a, p)))$$

Where d is the squared Euclidean distance (negative dot product can also be used instead). Note that it is sensitive to the scale of the image feature. In our implementation, we normalize the image feature to have unit magnitude and then scale it by 4, as suggested by [16]. An alternative is to not do normalization and let the network learn the scaling, as suggested [63]; which also works in our experience, though one has to be careful with the initialization to avoid numerical instability at the beginning.

Table 4.1: R@k performance on 3 benchmarks.

Dataset	Cars-196					
k	1	2	4	8	16	32
NetA:pool5.3	87.8	92.7	95.6	97.5	98.6	99.2
Dataset	CUB-200-2011					
k	1	2	4	8	16	32
NetA:pool5.3	66.4	77.5	85.4	91.3	95.2	97.1

Dataset	Product			
k	1	10	100	1000
NetA:pool5.3	74.8	88.3	95.2	98.5

Table 4.2: R@1 performance on 3 benchmarks, compared to previous works.

Method	Network	dim	Cars-196	CUB-200	Product
Lifted structure [82]	GoogLeNet	64/64/512	53.0	47.2	62.5
Facility [83]	Inceptionv1BN	64	58.1	48.2	67.0
Angular loss [71]	GoogLeNet	512	71.4	54.7	70.9
HDC [74]	GoogLeNet	128x3	73.7	53.6	69.5
BIER [73]	GoogLeNet	512	78.0	55.3	72.7
Margin [70]	ResNet50	128	<u>79.6</u>	**63.6**	72.7
Proxy-NCA [72]	InceptionBN	64	73.2	49.2	<u>73.7</u>
ABE [84]	8 Heads Ensemble	512	**85.2**	<u>60.6</u>	**76.3**
Our NetA	VGG16-BN (layer pool5.3)	512	**87.8**	**66.4**	**74.8**
	VGG16-BN (layer fc6)	512	81.0	61.7	<u>74.1</u>
Our NetE	VGG16-BN (layer pool5.3)	512	<u>86.7</u>	<u>65.8</u>	73.6
	VGG16-BN (layer fc6)	512	84.3	63.7	73.3
	VGG16-BN (layer fc7)	512	71.4	56.8	71.4

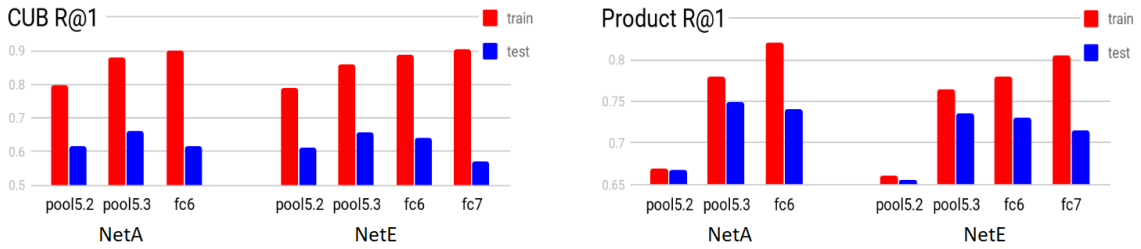

Figure 4.8: R@1 performance of NetA and NetE on CUB-200-2011 (left) and Stanford Online Product (right)

4.5 Experiments

Here we provide more detail about the experiment set up on 3 fine-grained image retrieval datasets: **Cars196** [75]: consist of 16k images of 196 different classes of cars. Following the typical protocol, we train on the first 98 classes and test on the other 98 classes. **CUB-200-2011** [85]: has around 12k images of 200 bird species. The first 100 classes (5864 images) are used for training and the rest for testing. **Stanford Online Product** [86]: is bigger than the other 2 with 120k images of more than 22k products from eBay. Similarly, the first half (11318 classes with a total of 59551 images) are used for training and the remaining classes are for testing.

We use stochastic gradient descent with momentum weight 0.9, weight decay factor 5e-4, learning rate 0.01 (decreasing by 10 times during training). Random scaling, cropping, and flipping are used for data augmentation; a single center crop is used during test time. We use batch size 32, train for 20k iterations in case of Cars-196 and CUB-200-2011, and 200k iterations in case of Online Product.

4.5.1 Ablation study

We switch the backbone architecture from VGG-16-batchnorm to Alexnet and Resnet34. As shown in Figure 4.6, the observation is still similar: the penultimate layer is better than output layer during test time; though the performance is lower compared to using VGG.

We lower the output dimensionality from 512 to 256, 128 and 64. As shown in Figure

Figure 4.9: Some nearest neighbor (NN) retrieval examples on the 3 datasets, we show cases in which using feature from different layers results in different NN.

4.7, this degrades the output layer fc6 performance, while the penultimate layer pool5.3 (still with dimensionality 512) is affected slightly.

The soft triplet based loss we used are popular and work well across different image retrieval tasks. In our experience, the contrastive loss, also popular, is slightly less effective. The classification based loss fits extremely well, but can be more vulnerable to overfitting, as shown in figure 4.3.

4.5.2 Result

In Figure 4.8-left, we show NetA and NetE performance on CUB-200-2011. This dataset is much harder to generalize even though its size is similar to that of Cars-196. Still, pool5.3 is the best performing layer, and fc6 does better if it is not next to the lost layer.

In Figure 4.8-right, we show performance on Stanford Online Product. Similar to previous cases, fc6 is outperformed by pool5.3; however due to this dataset's bigger size, there are some differences: (1) it takes much longer to train, (2) NetA seems to fit the training data slightly better/faster than NetE, resulting in better test time performance too. We speculate that when training at an even larger scale, bad generalization will be a less significant concern. We show some retrieval qualitative result in the supplemental.

Comparing to state-of-the-arts: in Table 4.2, we report the R@1 performance of different layers of NetA and NetE. As a baseline, our fc6 yields comparable result to some state of the art approaches, and our pool5.3 result outperform all of them. Note that each component (loss function, mining strategy, etc) of each system might not be directly comparable because of difference in network architecture and embedding dimension (and possibly data preprocessing, training hyper-parameters, etc). Moreover, we are investigating the generalization effect of different layers, not proposing a new method to replace existing works.

4.5.3 Analyzing publicly released models

This effect can also be demonstrated on other models, not just ours. Among previous works, LiftedStructure [82] and HDC [74] have released the models used in their papers. We obtained and tested these models, the result is shown in Table 4.3. Similar to our system, the output layers here are also outclassed by the layer before it during test time.

One could argue that, different from our case, it is mainly because of the bigger embedding size here. However notice that output layers still fit quite well on training data even with smaller embedding size, but generalize worse on testing data. [82] made an observation that the embedding size did not play a significant role in their system. But we do observe that the size is a contributing factor affecting both fitting and generalizing performance, and not only that of the embedding layer but also other layers' as well.

Finally, we observe that our system generalizes better: at similar training performance, our output layer still perform better on the test set (refer to Figure 4.4 and 4.8); this is also the case for penultimate layers. We speculate the reason to be the difference in embedding size, loss function, network architecture, and its corresponding pretrained model. Worse generalization might be what prevented previous works from fitting the training data further.

4.5.4 Experiment on revisited Oxford-5k and Paris-6k benchmarks

We perform additional experiments on 2 popular scene image retrieval datasets: Oxford [87] and Paris [88], using the new labels and benchmarking protocol introduced in [89].

Implementation detail: we use the NetA same as previous experiments, but with VGG-16 (without batch-norm) as the backbone architecture. We use the same training data as in [20], which we manage to collect around 20k images. During training, images are resized and cropped to 500x500.

In [89], 3 different evaluation setups are introduced: Easy (E), Medium (M) and Hard

Table 4.3: R@1 performance of different layers from publicly released lifted structure model. Penultimate denotes the layer before the output layer.

R@1	Cars-196		CUB-200		Product	
Layer	train	test	train	test	train	test
Lifted Structure released model [82]						
model-512	**97.7**	47.9	66.4	46.3	65.2	62.1
penultimate	97.0	**65.1**	66.4	**47.1**	**67.7**	**65.9**
HDC released model [74]						
outputs	**80.4**	73.7	**83.7**	53.6	**76.5**	69.5
penultimates	75.5	**75.7**	79.9	**57.2**	75.6	**70.1**
Our NetA						
output (fc6)	**92.4**	81.0	**89.6**	61.7	**81.2**	74.1
penultimate	90.4	**87.8**	87.6	**66.4**	77.9	**74.8**

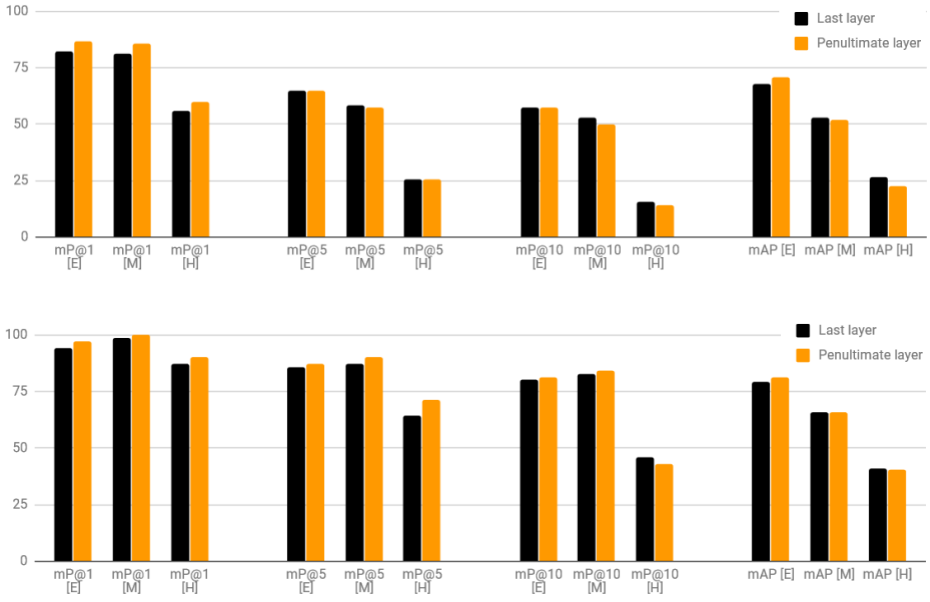

Figure 4.10: Image retrieval performance on ROxford-5k (top) and RParis-6k (bottom), when using last layer vs penultimate layer for feature extraction

(H), with several metrics: mP@1, mP@5, mP@10 and mAP (we refer readers to [89] for more detail). We show the result under these different setups in Figure 4.10 when using the last layer vs the penultimate layer.

It can be observed that using the last layer tends to get a worse result in Easy setup or using mP@1 metric, while having the advantage in Hard setup and using mP@10 or mAP metric. Note that mP@1 metric here is the same as the R@1 metric used in the 3 previous fine-grained image retrieval benchmarks, so in this case, it's consistent with previous observation that penultimate layer is better.

However under mAP metric, and especially in Hard mode, using the last layer is better. The explanation could be how differently these metrics reflect the result quality. When the application is to search for relevant reference images (for example "collect all Eiffel tower images in the database") mAP in Hard mode should be used; on the other hand, if the task is using image retrieval to make prediction about the query image (for example "identify this tower" or "recognizing this face"), R@k, R@1 or mP@1 in Easy mode is more appropriate. Our speculation is that, when using penultimate layer, examples of the same class form more complex manifold in this latent space, such that it's easier to search for few neighbors of the same classes, but more difficult to identify all instances of that class.

State of the art systems, with better training data collection procedure, additional bells and whistles, have achieved more impressive results on these benchmarks, we refer readers to [89] for detail.

4.5.5 Experiment on Crossview Image Retrieval

We perform additional experiment on the crossview image retrieval dataset (chapter 3). This task is cross domain, the goal is to learn a joint embedding feature space where streetview images and overhead view images are comparable (for example the streetview image and the overhead view image of the same scene should be close). In chapter 3, we

studied different DML approaches and also proposed orientation invariant training to improve retrieval performance on this dataset. Note that 2 different encoder (CNN) are leant for embedding streetview and crossview images as they are different kinds of images.

First we attempt using image features produced by the penultimate layer and other layers before it. Naturally this fails to work at all because the streetview embedding network and the overhead view embedding network only shares the feature space of the last output.

We reimplement the training pipeline with the following differences:

- Using PyTorch library and pretrained ResNet18 as backbone architecture, which is more powerful than our old caffe, Alexnet implementation.
- Sharing last layer strategy: this layer defines a linear transformation between the penultimate layer's feature and the output feature; by sharing it, the feature spaces of penultimate layers from the 2 network would become similar by design.
- Sharing all non-BatchNorm layers strategy: this attempts to make the 2 embedding network as similar as possible, the only difference is that BatchNorm parameters differ when running on different data input (here streetview images vs overhead view images).

After training for 150k iterations on the training set, we measure the Recall at 1% performance on Denver test set comparing last layer feature and penultimate layer feature; the result is shown in Figure 4.11. While sharing the whole CNN for encoding both streetview and overhead images is the best strategy to make penultimate layer feature work, it is still not as good as using last layer feature. We believe that even with full sharing, the inherent difference between 2 domains still make image features different, more so at lower level feature. For example, at the highest level feature (last layer), it could contain information about scene layout and objects like buildings and houses; while at low level, different features could be utilized to recognize houses such as doors and windows in the case of streetview images, and roofs in the case of overhead images.

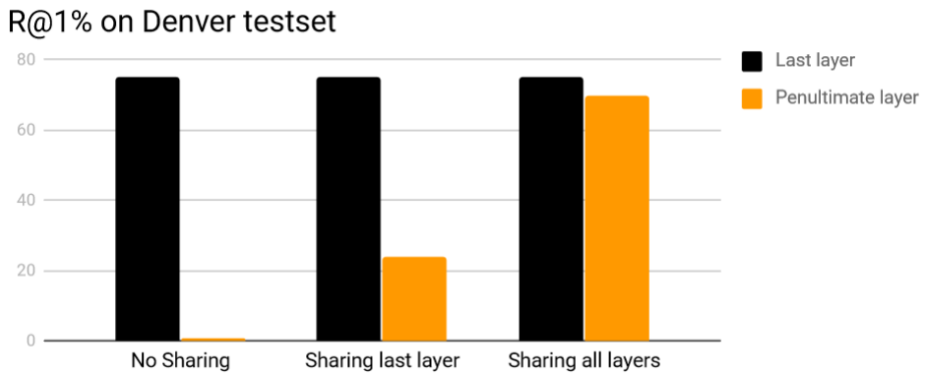

Figure 4.11: Crossview image retrieval performance comparing features from different layers and different training strategies.

4.6 Conclusion

We analyze generalization as a big contributing factor to improvements made on small scale fine-grained image retrieval benchmarks. As we have shown, choosing a different loss or different feature extraction layer can improve test time performance even though it actually under-perform on training data, and vice versa. Hence generalization should be kept in mind when designing or analyzing new techniques in the future.

Given that any layer in a network can be used for feature extraction, the major part of this work is an ablation study of how each layer performs differently in the fine-grained image retrieval task. Not too surprising, we found that the closer a layer to the loss layer, the better its produced feature is. Different from what people would assume, the last layer is usually however not the best one at test time, due to bad generalization; the second last layer is. The observation holds across different benchmarks and not only on our model, but also other models released by previous works.

We also demonstrate use cases in which the practice does not work well. First is in the large scale data or challenging training, in which overfitting is not a concern. Second is the case of joint embedding across multiple different domains.

CHAPTER 5
DEEP METRIC LEARNING FOR IM2GPS

5.1 Introduction

Learning to localize images with GPS labels under Deep Metric Learning (DML) approach is very challenging. As shown in chapter 2, we can train a network using the classification approach and use it to perform geolocalization either by classification, or by image retrieval method; however the ranking network finetuned with DML approach failed to deliver noticeable improvement compared to simply using the classification network as feature extractor.

DML is usually very sensitive to label noise and slow to converge on data that is too diverged. Recall that in DML approach, there is not a fixed "target label" for each training image; a DML loss function operates based on interaction between multiple images (pair, triplet or cluster), hence the noise/diversity is multiplied. Successful previous works on scene image retrieval relies on curated image dataset: usually images in the same location have to go through geometry verification process to make sure that they are of exact same scene before being used as source of supervision for DML training [20, 19].

Recent works on Im2GPS geolocalization [90, 91] have demonstrated new impressive result. Both use more powerful backgone architectures to significantly improve the geolocalization accuracy; for example at street level, switching from VGG to ResNet101 nearly double the accuracy.

In this chapter, we show that it is possible to train an image retrieval system for geolocalization application with DML, by using more powerful backbone architecture and pretraining the system on classification task. The obtained network is more suitable for image retrieval use case.

5.2 Approach

We briefly outline the Deep Metric Learning system we are gonna experiment with. Note that this formulation has been successfully applied to crossview image retrieval task and fine-grained image retrieval task, as shown in previous chapters.

5.2.1 Revisit DML loss function and the exhausting mini-batch trick

All DML loss functions have similar objective: to make images labeled similar close, and images labeled dissimilar far from each other in the feature space. We will focus on the soft triplet loss in particular (which is proposed in chapter 3 and successfully used in [63]). Given a triplet: an anchor image feature a, a positive image feature p and a negative image feature n, the loss function is defined as:

$$f_{triplet}(a,p,n) = -log(p_{triplet}(a,p,n)) \tag{5.1}$$

$$p_{triplet}(a,p,n) = \frac{exp(s(a,p))}{exp(s(a,p)) + exp(s(a,n))}$$

Where $s()$ is the similarity function, in our implementation it is negative L2 distance between 2 vectors. Here $p_{triplet}(a,p,n)$ can be interpreted as the probability that a is more similar to p than to n, and $f_{triplet}$ is the cross entropy or negative log likelihood function.

A good strategy to make use of it, as explained in chapter 3 is exhausting mini-batch training: (1) sample a batch of images, from which many triplets can be formed, then (2) actually apply the loss function to all possible triplets in that batch and use the averaged loss value as training objective function.

We also experiment with another version of the loss function, where the triplet based loss becomes the function of an anchor image feature a, a set of positive image features p and a set of negative image features n:

$$f_{triplet}(a,p,n) = -log(p_{triplet}(a,p,n)) \tag{5.2}$$

$$p_{triplet}(a,p,n) = \frac{\sum_i exp(s(a,p_i))}{\sum_i exp(s(a,p_i)) + \sum_j exp(s(a,n_j))}$$

Compared to the old loss, there is 2 adjustments being made: (1) More discriminative: with multiple negative examples; jointly comparing the anchor against multiple negative examples at the same time helps fitting the data faster, since the loss function can now prioritize the harder negative examples; and (2) More tolerable to diverged data: with multiple positive examples; our data is very diverged, our geolocalization task essentially aim to "retrieval a correct example", not "retrieval all correct examples"; Having multiple positive examples reflects the task objective more closely.

5.2.2 Increase mini-batch size and large batch size training strategy

Algorithm 1 Back propagation algorithm with big mini-batch

Input: network being trained f_θ, loss function L, mini-batch of images b and labels l
Split the mini-batch: $b_1, b_2, ...b_m = split(b)$
Forward pass on each one on GPU and save the result to CPU/RAM: $o_i = f_\theta(b_i)$
    Combine the output $o = combine(o_0, o_1, ...o_m)$
Compute training loss with $L(o,l)$
    Compute output gradient and save the result to CPU/RAM: $\nabla o = \frac{d}{do}L(o,l)$
    Split the output's gradient: $\nabla o_1, \nabla o_2, ...\nabla o_m = split(\nabla o)$
We want $\nabla\theta = \frac{d}{d\theta}L(o,l) = \sum_{i=1}^{m} \frac{d}{do_i}L(o,l)\frac{d}{d\theta}o_i$
    Initialize: $\nabla\theta = 0$
    Forward and backward on each one on GPU and accumulate gradient:
        Redo forward pass $o_i = f_\theta(b_i)$
        From ∇o_i, perform backward pass $\nabla\theta = \nabla\theta + \nabla o_i \frac{d}{d\theta}o_i$
Output: Parameter gradient $\nabla\theta$

Training with large large mini-batch might be beneficial because of more stable gradient. Moreover, with exhausting mini-batch trick, we are able to increase the number of triplets quadratically or cubically with respect to mini-batch size. However increasing mini-batch size in gradient decent training is not trivial when we are relying on GPUs with limited memory. Note that different from traditional losses which operates at image level: a function of an image and its corresponding label (computing the loss over a mini-batch is simply the average loss over each image in the batch), our DML loss operates at mini-

batch level: it is a function of all images in the mini-batch and their labels; hence increasing mini-batch size by accumulating gradient over multiple iteration (instead of 1 only) does not work.

We adjusted the back propagation algorithm to work with big mini-batch, shown in. The procedure is (1) split the mini-batch to fit in the GPUs when doing the forward pass and releasing GPU memory immediately, then recombine to have the full output; (2) compute the loss and the gradient with respect to the output; (3) finally split the mini-batch and do forward again followed by a backward pass from the computed output's gradient. Since this procedure actually do forward pass twice, it is 1.5 time slower than the traditional forward backward algorithm.

5.3 Experiment

The experiment set up is the same as what has been done in chapter 2. We use the Im2GPS dataset for training, consisting of around 6 million gps-tagged images. The same testset Im2GPS3k and YFCC4k will be used for comparison and analysis.

The same metric will be used: geolocalization accuracy at 5 different threshold levels: street (1km), city (25km), region (200km), country (750km) and continent (2500km). If the predicted GPS location is within the threshold distance from the ground-truth GPS, it'll be consider correct localization at that threshold level.

5.3.1 Implementation Detail

We reimplement the classification training pipeline from chapter 2 with 2 changes: (1) use more fine grained GPS quantization schemes with around 17k classes (2) use more powerful backbone architecture ResNet50. We will refer to this model as [L3] to differentiate it from the VGG model [L] and [L2] we previous trained (section 2.4).

We then finetune [L3] with DML training. We experiment with 3 different settings, resulting in 3 ranking networks that can be used for image retrieval:

Table 5.1: Performance on Im2GPS3k

Threshold (km)		Street 1	City 25	Region 200	Country 750	Cont. 2500
[L] Classification	VGG16 + 7k gps classes	4.0	14.8	21.4	32.6	52.4
[L] 1NN Retrieval	6 million images database	7.5	18.9	23.5	32.6	49.5
[R] 1NN Retrieval	6 million images database	7.5	18.8	24.0	33.2	49.6
[L3] Classification	ResNet50 + 17k gps classes	10.4	27.1	36.1	49.2	66.0
[L3] 1NN Retrieval	6 million images database	10.0	26.7	34.2	45.6	60.8
[R1] 1NN Retrieval	6 million images database	10.7	27.4	34.8	48.1	63.8
[R1] kNN	6 million images database	12.7	29.6	36.4	49.2	65.3
[R2] 1NN Retrieval	6 million images database	11.1	27.6	35.2	48.5	64.7
[R2] kNN	6 million images database	12.5	29.7	37.5	50.4	66.2
[R3] 1NN Retrieval	6 million images database	9.9	27.0	35.1	47.5	64.6
[R3] kNN	6 million images database	13.0	29.6	37.2	50.0	66.0
CPlaNet [90]		10.2	26.5	34.6	48.6	64.6
ISN [91]		10.5	28.0	36.6	49.7	66.0

- R1: Loss function 5.1 is used. A mini-batch of size 32 is randomly sampled at each iteration as following: 8 randomly (gps quantized) classes are randomly chosen, then 4 images are sampled from each class.
- R2: similar to R1, but batch size is set at 256.
- R3: we use loss function 5.2; batch size is set at 2048 consisting of 64 classes with 32 randomly sampled examples per class (hence for each anchor there is 31 positives and 2016 negative examples).

5.3.2 Result and Analysis

The geolocalization accuracy on 2 testsets are shown in Table 5.1 and 5.2.

All the new models we trained with better a backbone architecture significantly outperform the previous ones. At street level, classification with [L3] yields double accuracy compared to [L]. These result are consistent and comparable with [90, 91].

However using [L3] in NN retrieval approach actually result in worse performance than classification approach, unlike model [L]. It seems more powerful architecture is greatly

Table 5.2: Performance on YFCC4k

Threshold (km)		Street 1	City 25	Region 200	Country 750	Cont. 2500
[L] Classification	VGG16 + 7k gps classes	1.6	4.9	8.6	19.3	37.4
[L] 1NN Retrieval	6 million images database	2.1	5.2	7.9	17.6	36.0
[R] 1NN Retrieval	6 million images database	2.4	5.5	8.3	18.3	37.2
[L3] Classification	ResNet50 + 17k gps classes	3.6	9.3	15.0	28.3	48.8
[L3] 1NN Retrieval	6 million images database	3.7	8.9	13.1	25.6	45.1
[R1] 1NN Retrieval	6 million images database	4.2	9.4	14.6	28.4	49.3
[R1] kNN	6 million images database	4.8	10.3	15.6	29.0	49.9
[R2] 1NN Retrieval	6 million images database	4.1	9.2	14.0	27.3	47.8
[R2] kNN	6 million images database	4.7	10.6	15.6	29.3	49.5
[R3] 1NN Retrieval	6 million images database	3.7	9.1	14.1	27.4	48.9
[R3] kNN	6 million images database	4.7	10.4	15.3	28.5	50.0
CPlaNet [90]		7.9	14.8	21.9	36.4	55.5

Table 5.3: Performance on Im2GPS testset

Threshold (km)		Street 1	City 25	Region 200	Country 750	Cont. 2500
[L3] Classification	ResNet50 + 17k gps classes	12.2	43.0	52.7	67.1	79.7
[R2] kNN	6 million images database	18.6	43.5	51.1	65.4	79.3
CPlaNet [90]		16.5	37.1	46.4	62.0	78.5
ISN [91]		16.9	43.0	51.9	66.7	80.2

beneficial for classification approach, more so than for retrieval approach. Note that retrieval approach can alternatively benefit from bigger retrieval database.

Finally retrieval approach using DML model [R1], [R2], [R3] all outperform using model [L3]. This shows that DML fine-tuning does improve the model quality as feature extractor for image retrieval set up. Similar to experiment in section 2.4, switching from NN to kNN smoothing gives extra boost in accuracy. Among [R2] seems slightly better though the difference between 3 models are not significant.

5.3.3 Comparing to state of the art

We compare to recent result from [90, 91] on 3 testsets in Table 5.1, 5.2 and 5.3. On Im2GPS3k and Im2GPS testset we have comparable or better accuracy.

On the YFCC4k test set, our performance is much inferior to that of [90], even when using the same classification method. This is due to our training data Im2GPS being biased and not reflecting the distribution of Flickr images.

5.4 Discussion

While applying DML to noisy or diverged data could be difficult, we show that it could work with powerful backbone architecture. We experiment with training several models, taking advantage of recent DML techniques: the DBL loss for triplet, exhausting mini-batch, large batch size training and modified triplet loss for diverged data. When using for image retrieval and geolocalization, all of our DML trained models are better then the classification trained model. State of the art performance are demonstrated on Im2GPS benchmarks.

CHAPTER 6
CONCLUSION

In this thesis, we study image localization task and explore image ranking/retrieval approach. Deep Learning has advanced many computer vision task including image retrieval; in addition, images data has become increasingly abundant. This has translated to great progress in image retrieval and localization.

Our first contribution is a study of image geolocalization at planet scale (Im2GPS: predicting GPS coordinate from image data) comparing 2 deep learning approaches: image classification and image retrieval. Image classification aims to train a deep network that classifies images into one of multiple regions, each associated with a GPS coordinate. On the other hand, image retrieval approach uses deep network to extract meaningful features; an input image's feature is matched against a database of GPS tagged reference images to make a location prediction. Our study discovers trade off between localization accuracy at different granularity levels. Image retrieval approach has great advantage when it comes to geolocalization at fine levels (street, city) and still competitive at coarse levels (country, continent).

Next, we investigate different architectures for matching and retrieving crossview images. The application is to do localization using image retrieval approach where the query images are normal streetview images, but the reference images in database are overhead viewpoint (satellite images); we collect and publish a large scale dataset for this task. Extensive experiment is performed to compare different architectures (Siamese vs Triplet) and different metric learning loss (Contrastive vs Triplet Hinge vs our proposed DBL loss). Last but not least, we show how to further improve retrieval performance by incorporating supervised alignment information to learn a rotational invariant representation.

Our third contribution is exploring state of the art Deep Metric Learning (DML) tech-

niques in image retrieval. We first look at it in the context of fine grained image retrieval, which is much well studied and many progresses have been recently made in the literature. We discover that, while in many DML systems, the embedding layer is always the last one in the network, it might not be the optimal one; previous DML works have not investigated which layer in a network is best suited for feature extraction. Our ablation study shows that the last layer perform the best during training, but can be vulnerable to overfitting; during test time, the penultimate layer usually outperforms the last one. We make use of that as a regularization practice and demonstrate state of the art performance on 3 fine grained image retrieval benchmarks.

Lastly, we apply DML techniques to training deep network for image retrieval and Im2GPS geolocalization task. Our experiment shows that DML trained systems outperform a classification trained system as feature extractors, resulting in better image retrieval and geolocalization performance.

REFERENCES

[1] J. Hays, A. Efros, *et al.*, "Im2gps: Estimating geographic information from a single image," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, 2008, pp. 1–8.

[2] T. Weyand, I. Kostrikov, and J. Philbin, "Planet-photo geolocation with convolutional neural networks," in *European Conference on Computer Vision*, Springer, 2016, pp. 37–55.

[3] Y. Li, D. J. Crandall, and D. P. Huttenlocher, "Landmark classification in large-scale image collections," in *Computer vision, 2009 IEEE 12th international conference on*, IEEE, 2009, pp. 1957–1964.

[4] J. Hays and A. A. Efros, "Large-scale image geolocalization," in *Multimodal Location Estimation of Videos and Images*, Springer, 2015, pp. 41–62.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in Neural Information Processing Systems*, 2014, pp. 487–495.

[7] A. Dubey, N. Naik, D. Parikh, R. Raskar, and C. A. Hidalgo, "Deep learning the city: Quantifying urban perception at a global scale," in *European Conference on Computer Vision*, Springer, 2016, pp. 196–212.

[8] S. Lee, H. Zhang, and D. J. Crandall, "Predicting geo-informative attributes in large-scale image collections using convolutional neural networks," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, IEEE, 2015, pp. 550–557.

[9] S. Bell and K. Bala, "Learning visual similarity for product design with convolutional neural networks," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 98, 2015.

[10] J. Huang, R. S. Feris, Q. Chen, and S. Yan, "Cross-domain image retrieval with a dual attribute-aware ranking network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1062–1070.

[11] P. Sangkloy, N. Burnell, C. Ham, and J. Hays, "The sketchy database: Learning to retrieve badly drawn bunnies," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 119, 2016.

[12] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, IEEE, 2014, pp. 1701–1708.

[13] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[14] T.-Y. Lin, Y. Cui, S. Belongie, and J. Hays, "Learning deep representations for ground-to-aerial geolocalization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5007–5015.

[15] S. Workman, R. Souvenir, and N. Jacobs, "Wide-area image geolocalization with aerial reference imagery," in *ICCV 2015*, 2015.

[16] N. N. Vo and J. Hays, "Localizing and orienting street views using overhead imagery," in *European Conference on Computer Vision*, Springer, 2016, pp. 494–509.

[17] G. Tolias, R. Sicre, and H. Jégou, "Particular object retrieval with integral max-pooling of cnn activations," *ArXiv preprint arXiv:1511.05879*, 2015.

[18] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5297–5307.

[19] F. Radenović, G. Tolias, and O. Chum, "Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples," in *European Conference on Computer Vision*, Springer, 2016, pp. 3–20.

[20] A. Gordo, J. Almazán, J. Revaud, and D. Larlus, "Deep image retrieval: Learning global representations for image search," in *European Conference on Computer Vision*, Springer, 2016, pp. 241–257.

[21] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, IEEE, vol. 2, 2006, pp. 1735–1742.

[22] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *The Journal of Machine Learning Research*, vol. 11, pp. 1109–1135, 2010.

[23] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in neural information processing systems*, 2012, pp. 1061–1069.

[24] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, IEEE, 2014, pp. 1386–1393.

[25] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev, "Metric learning with adaptive density discrimination," *ArXiv preprint arXiv:1511.05939*, 2015.

[26] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[27] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, IEEE, 2007, pp. 1–8.

[28] ——, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, 2008, pp. 1–8.

[29] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European conference on computer vision*, Springer, 2008, pp. 304–317.

[30] A. R. Zamir and M. Shah, "Accurate image localization based on google maps street view," in *Computer Vision–ECCV 2010*, Springer, 2010, pp. 255–268.

[31] H. Altwaijry, E. Trulls, J. Hays, P. Fua, and S. Belongie, "Learning to match aerial images with deep attentive architectures," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[32] T.-Y. Lin, S. Belongie, and J. Hays, "Cross-view image geolocalization," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, IEEE, 2013, pp. 891–898.

[33] A. R. Zamir, A. Hakeem, L. Van Gool, M. Shah, and R. Szeliski, "Large-scale visual geo-localization," *Advances in computer vision and pattern recognition (*, 2016.

[34] M. Bansal, K. Daniilidis, and H. Sawhney, "Ultra-wide baseline facade matching for geo-localization," in *Computer Vision–ECCV 2012. Workshops and Demonstrations*, Springer, 2012, pp. 175–186.

[35] Q. Shan, C. Wu, B. Curless, Y. Furukawa, C. Hernandez, and S. M. Seitz, "Accurate geo-registration by ground-to-aerial image matching," in *3D Vision (3DV), 2014 2nd International Conference on*, IEEE, vol. 1, 2014, pp. 525–532.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ArXiv preprint arXiv:1409.1556*, 2014.

[37] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li, "Yfcc100m: The new data in multimedia research," *Communications of the ACM*, vol. 59, no. 2, pp. 64–73, 2016.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[39] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *ArXiv preprint arXiv:1408.5093*, 2014.

[41] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, IEEE, vol. 1, 2005, pp. 539–546.

[42] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a siamesei time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[43] T. Weyand, I. Kostrikov, and J. Philbin, "Planet - photo geolocation with convolutional neural networks," *CoRR*, vol. abs/1602.05314, 2016.

[44] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[45] F. Wang, L. Kang, and Y. Li, "Sketch-based 3d shape retrieval using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1875–1883.

[46] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 37–45.

[47] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg, "Matchnet: Unifying feature and metric learning for patch-based matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3279–3286.

[48] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao, "Online multimodal deep similarity learning with application to image retrieval," in *Proceedings of the 21st ACM international conference on Multimedia*, ACM, 2013, pp. 153–162.

[49] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," *Proceedings of the British Machine Vision*, vol. 1, no. 3, p. 6, 2015.

[50] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2794–2802.

[51] P. Sangkloy, N. Burnell, C. Ham, and J. Hays, "The sketchy database: Learning to retrieve badly drawn bunnies," *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

[52] Q. Ke and Y. Li, "Is rotation a nuisance in shape recognition?" In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 4146–4153.

[53] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng, "Tiled convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2010, pp. 1279–1287.

[54] R. Gens and P. M. Domingos, "Deep symmetry networks," in *Advances in neural information processing systems*, 2014, pp. 2537–2545.

[55] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction," *Monthly Notices of the Royal Astronomical Society*, vol. 450, no. 2, pp. 1441–1459, 2015.

[56] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in *Advances in Neural Information Processing Systems*, 2014, pp. 1988–1996.

[57] J. Lin, O. Morere, V. Chandrasekhar, A. Veillard, and H. Goh, "Deephash: Getting regularization, depth and fine-tuning right," *ArXiv preprint arXiv:1501.04711*, 2015.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[59] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014*, Springer, 2014, pp. 818–833.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[61] N. Vo, N. Jacobs, and J. Hays, "Revisiting im2gps in the deep learning era," *ICCV*, 2017.

[62] S. Ding, L. Lin, G. Wang, and H. Chao, "Deep feature learning with relative distance comparison for person re-identification," *Pattern Recognition*, vol. 48, no. 10, pp. 2993–3003, 2015.

[63] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *ArXiv preprint arXiv:1703.07737*, 2017.

[64] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *British Machine Vision Conference*, 2015.

[65] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, 1994, pp. 737–744.

[66] O. Tadmor, Y. Wexler, T. Rosenwein, S. Shalev-Shwartz, and A. Shashua, "Learning a metric embedding for face recognition using the multibatch method," *ArXiv preprint arXiv:1605.07270*, 2016.

[67] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Advances in Neural Information Processing Systems*, 2016, pp. 1849–1857.

[68] V. B. Kumar, B. Harwood, G. Carneiro, I. Reid, and T. Drummond, "Smart mining for deep metric learning," *ArXiv preprint arXiv:1704.01285*, 2017.

[69] C. Huang, C. C. Loy, and X. Tang, "Local similarity-aware deep feature embedding," in *Advances in Neural Information Processing Systems*, 2016, pp. 1262–1270.

[70] C.-Y. Wu, R Manmatha, A. J. Smola, and P. Krähenbühl, "Sampling matters in deep embedding learning," *ArXiv preprint arXiv:1706.07567*, 2017.

[71] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," *ArXiv preprint arXiv:1708.01682*, 2017.

[72] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh, "No fuss distance metric learning using proxies," *ArXiv preprint arXiv:1703.07464*, 2017.

[73] M. Opitz, G. Waltner, H. Possegger, and H. Bischof, "Bier-boosting independent embeddings robustly," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5189–5198.

[74] Y. Yuan, K. Yang, and C. Zhang, "Hard-aware deeply cascaded embedding," *ArXiv preprint arXiv:1611.05720*, 2016.

[75] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[76] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *European Conference on Computer Vision*, Springer, 2016, pp. 499–515.

[77] S. Horiguchi, D. Ikami, and K. Aizawa, "Significance of softmax-based features in comparison to distance metric learning-based features," *ArXiv preprint arXiv:1712.10151*, 2017.

[78] F. Radenović, G. Tolias, and O. Chum, "Fine-tuning cnn image retrieval with no human annotation," *ArXiv preprint arXiv:1711.02512*, 2017.

[79] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[80] K. Greff, R. K. Srivastava, and J. Schmidhuber, "Highway and residual networks learn unrolled iterative estimation," *ArXiv preprint arXiv:1612.07771*, 2016.

[81] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *ArXiv preprint arXiv:1708.02002*, 2017.

[82] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4004–4012.

[83] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy, "Deep metric learning via facility location," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[84] W. Kim, B. Goyal, K. Chawla, J. Lee, and K. Kwon, "Attention-based ensemble for deep metric learning," *European Conference on Computer Vision*, 2018.

[85] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," Tech. Rep., 2011.

[86] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[87] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[88] ——, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[89] F. Radenović, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Revisiting oxford and paris: Large-scale image retrieval benchmarking," in *CVPR*, 2018.

[90] P. H. Seo, T. Weyand, J. Sim, and B. Han, "Cplanet: Enhancing image geolocalization by combinatorial partitioning of maps," in *European Conference on Computer Vision*, Springer, 2018, pp. 544–560.

[91] E. Müller-Budack, K. Pustu-Iren, and R. Ewerth, "Geolocation estimation of photos using a hierarchical model and scene classification," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 563–579.